AD-A265 470



(2)

RL-TR-92-345, Vol IV (of seven)
Final Technical Report
December 1992

DTIC
ELECTE
JUN 9 1993
S C D



SYSTEM ENGINEERING CONCEPT DEMONSTRATION, Interface Standards Studies

Software Productivity Solutions, Inc.

J. Kaye Grau, Edward R. Comer, Sharon L. Rohde

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Copyright 1992 Software Productivity Solutions, Inc.
This material may be reproduced by or for the U.S. Government pursuant to the copyright license under clause at DFARS 252.227-7013 (October 1988).

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

93-12880



93 6 03 028

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-92-345, Volume IV (of seven) has been reviewed and is approved for publication.

APPROVED: Frank A LaMonica

FRANK S. LAMONICA
Project Engineer

FOR THE COMMANDER: John A Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1992	Final Feb 90 - Jul 92

4. TITLE AND SUBTITLE
SYSTEM ENGINEERING CONCEPT DEMONSTRATION,
Interface Standards Studies

5. FUNDING NUMBERS
C - F30602-90-C-0021
PE - 527027
PR - 5581
TA - 19
WU - 54

6. AUTHOR(S)
J. Kaye Grau, Edward R. Comer, Sharon L. Rohde

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Productivity Solutions, Inc.
122 4th Avenue
Indialantic FL 32903-1697

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3CB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-92-345, Vol IV
(of seven)

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Frank S. LaMonica/(315) 330-2054

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This final technical report documents the objectives, activities, and results of Air Force contract F30602-90-C-0021, entitled "System Engineering Concept Demonstration." The effort, which was conducted by Software Productivity Solutions, Inc., with McDonnell Douglas Corporation - Douglas Aircraft Company and MTM Engineering Inc. as subcontractors, demonstrated and documented the concept of an advanced computer-based environment which provides automation for Systems Engineering tasks and activities within the Air Force computer-based systems life cycle. The report consists of seven (7) volumes as follows: I) Effort Summary, II) Systems Engineering Needs, III) Process Model, IV) Interface Standards Studies, V) Technology Assessments, VI) Trade Studies, and VII) Security Study.

This Volume (Volume IV - Interface Standards Studies), provides an evaluation of existing interface standards in the areas of computer-based environment frameworks, tools, and virtual machines that have potential for use in developing the envisioned systems engineering automation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
System Engineering, System Life Cycle Tools, System Life Cycle Environment	154
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

Table of Contents i

List of Figures v

List of Tables vi

Introduction 1

1. Interface Terminology 1

1.1. Definition of Terms 3

1.2. Views of an Automated Environment 4

1.3. Classification of Interfaces 6

1.4. Interaction of Interface Classes 7

2. Interface Models 8

2.1. Model of the Frameworks Interface 8

2.1.1. Model of User Interfaces 9

2.1.2. Model of Communications 11

2.1.3. Model of Repositories 15

2.2. Model of the Tool Interface 20

2.3. Model of the Virtual Machine Interface 23

3. Criteria for Evaluation of Interface Standards 24

4. Evaluation of Interface Standards 25

4.1 User Interfaces 25

4.1.1 GUI 25

4.1.2 The X Window System™ 26

4.1.3 OSF/Motif™ 28

4.1.4 Open Look™ 30

4.1.5 GKS 31

4.1.6 Microsoft Windows 34

4.2 Communications 35

4.2.1 GOSIP 35

4.2.2 TCP/IP 37

4.2.3 NFS 38

4.3 Repositories 39
4.3.1 Standardization Groups 39
4.3.1.1 OMG 39
4.3.1.2 OODBTG 42
4.3.2 Existing Standards and Standard Work in Progress 43
4.3.2.1 ATIS (Atherton Tool Integration Services) 43
4.3.2.2 ANSI X3.138 46
4.3.2.3 IRDS 47
4.3.2.4 PCTE and PCTE+ 51
4.3.2.5 CIS 54
4.3.2.6 PCIS 55
4.3.2.7 CAIS-A 56
4.3.2.8 SQL 57
4.3.3 Products 59
4.3.3.1 AD/Cycle 59
4.3.3.2 Network Database Language 62
4.3.3.3 CODASYL 63
4.3.3.4 Cohesion 66
4.3.3.5 Rational 67
4.3.3.6 Atherton Technology Product Line 69
4.3.3.7 SLCSE Database 70
5 Evaluation of Tool Interface Standards 73
5.1 Tool Standardization Efforts by CFI 73
5.2 Tool Interoperability 73
5.2.1 CEF 73
5.2.2 Link Database 74
5.2.3 Clipboard 78
5.2.4 Pipes 79
5.2.5 Object Request Broker 80
5.3 Data Interchange Formats 83
5.3.1 PDES/STEP 83
5.3.2 CDIF 86
5.3.3 ODA/ODIF 87
5.3.4 TIFF 89
5.3.5 PHIGS 91

5.4 Language Representation 96
5.4.1 PostScript 96
5.4.2 EPS 101
5.4.3 IDL 102
5.5 Document Representation with SGML 106

6 Virtual Machine Interface Standards 107
6.1 POSIX 107
6.2 UNIX™ 117

7. Updates to Interface Standards Studies 121
7.1. GOSIP 121
7.2. OMG and the Object Request Broker 122
7.3 PCTE and PCTE+ 124
7.4 PCIS 124
7.5. Rational 124
7.6. PDES/STEP 125
7.7 POSIX 125

8. Conclusions from Interface Standards Studies 126
8.1 The Frameworks Interface 127
8.1.1 User Interfaces 127
8.1.2 Communications 128
8.1.3 Repositories 128
8.2 The Tools Interface 130
8.3 The Virtual Machine Interface 131

9. Future R&D for Interface Standards 132

Accesion For
NTIS CRA&I ☒
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

List of Figures

Figure 1.2-1. External View of the Automated Environment 4
Figure 1.2-2. Internal View of the Catalyst Environment 6
Figure 1.4-1. Relationship of Interfaces in the Catalyst Environment 8
Figure 2.1.1-1. Layered Structure for Human-Computer Dialogue [SIS86] 10
Figure 2.2-1. Model of a Tool 20
Figure 2.2-2. Model of Tool to Tool Interaction 21
Figure 2.3-1. Operating Systems Function [KOC84] 23
Figure 4.3.2.1-1. ATIS Layers [BEY90] 44
Figure 5.2.5-1. Object Management Architecture Overview [SOL90] 81

List of Tables

Table 2.1.3-1. Reference Model of Information Layers 19
Table 4.3.3.3-1 Summary of DBTG Data Constructs 64
Table 5.4.3-1 IDL Language Features 104
Table 6.1-1 Applications Portability Profile [USA90] 111

Introduction

This is the fourth volume of the Final Technical Report (FTR) for the System Engineering Concept Demonstration (SECD), contract F30601-90-C-0021 sponsored by Air Force Rome Laboratory (RL). This document contains background information about interface terminology, models, criteria and evaluations of the standards. The organization of this document's discussion is as follows:

- Interface Terminology
- Interface Models
- Frameworks Interface Standards
 - User interfaces
 - Communications
 - Repositories
- Tool Interface Standards
- Virtual Machine Interface Standards

The document concludes with recommendations from each category of interface standards. The automated system engineering environment will incorporate these standards. These evaluations were used as drivers for the definition of requirements, design, and interface documents for Catalyst.

This document was originally written in 1990 and 1991 to evaluate the "state" of the standards in the area of interfaces. The descriptions of the various interface standards and their related hardware and software are current as of that time period. Section 7, Updates to Interface Standards Studies, was added in May 1992.

1. Interface Terminology

The goal of analyzing interface standards was to investigate and assess the potential use of current and emerging interface standards for the systems engineering environment. In order to reach this goal, we found it necessary to identify the organizations involved in formalizing standards. Based on a literature search, we evaluated a number of existing and emerging interface standards to determine the maturity and relevance of each interface standard. Included in the search were the following categories of standards:

- DoD Standards

- Military Services Standards
- CALS (Computer-Aided Acquisition and Logistics Support)
- NASA Standards
- ANSI Standards
- ISO Standards
- IEEE Standards
- NIST Standards
- FAA Standards
- De facto Standards
- Consortium backed Standards

Because many of the standardization processes overlap, organizations which endorse these standards must work together. Consequently, categories are not always clear cut. For example, Ethernet is categorized as an IEEE standard (IEEE802) and as a consortium backed standard (Xerox, DEC, Intel).

Within the federal government, many computer standards stem from the National Computer Systems Laboratory (NCSL, formerly ICST) of the National Institute for Standards and Technology (NIST, formerly NBS). NCSL's objectives are to increase productivity in both the government and the private sector and to contribute to the U.S. industries posture in the international marketplace.

The speed and visibility with which the computer industry standards (such as PHIGS+[1]) have developed show an increased sophistication in the development of standards. This change in the standardization process is a sign the industry is maturing. The future promises to bring us even more surprises and, as these standards gain in popularity, the stereotype may shift to "if it is based on a standard, it must be fast" [BUC90].

In addition to identifying the players in the standards games, we found it necessary to define terms of the Interface domain and construct views and models of the problem space. These definitions allowed us to understand how interface standards apply to systems. With this knowledge we can

[1] PHIGS+ is the Programmer's Hierarchical Interface Graphics with surface rendering extensions.

evaluate the maturity of interface standards for an automated system engineering environment in the next 5-7 years.

1.1. Definition of Terms

To begin to resolve the issue, we must first establish a meaning for the two words "standard" and "interface."

> A standard is a technical specification or other document available to the public, drawn up with cooperation and consensus or general approval of all interests affected by it, based on the consolidated results of science, technology and experience aimed at the promotion of optimum community benefits and approved by a standardization body. Standards also denote guidelines, codes of good practice conventions, and de facto standards [NAS86].

> An interface is a shared boundary over which data passes between components of a system (or systems) [NAS86].

In addition to the traditional definitions of a standard, several other factors play a part in the definition of a standard. For example, what appeals as a standard today is its ability to lower the cost of development for vendors and the cost of ownership (including training and support) for users. A standard need is to provide users with vendor independence and to promote applications that can interoperate across heterogeneous, networked systems in multinational, multilingual environments. In areas like networking and applications interoperability, standardization is a requirement (that is, no standards, no networking, no interoperability). Standards provide users with a measure of vendor independence that protects their software investment. Proactive adoption of standards to protect user's software investment is a profitable business, and today, these benefits are well understood. All these factors play a part in the definition of a standard and play a part in ensuring standard's success in the marketplace of computer industry.

Inherent in the definition of an interface is communication by the flow of data. To successfully communicate system data across the shared boundary of any two system components involves a mutual understanding of both major facets of the interface:

- The process or mechanism for information transfer, often referred to as the protocol
- The information itself that is being transferred

In the following sections, we address both the process and information involved with interfaces. We also develop a classification of the interfaces

and a model of the required interface interaction for information and data transfer in the automated systems engineering environment.

1.2. Views of an Automated Environment

In order to define the interfaces addressed by this effort, we developed two views of an automated environment: an external view and an internal view. These views explored two levels of abstraction of the systems engineering environment. Figure 1.2-1 illustrates an external or top level view of the automated environment. As shown in this figure, three components or objects were identified:

- The User(s)
- The Computer System Workspace which contains the Catalyst Environment
- The External Systems (i.e., other External Computer System Workspace(s) and/or other types of systems in the systems engineering environment)

As illustrated in Figure 1.2-1, the defined workstation and environment for the systems engineer is the Computer System Workspace. The arrows in this figure represent the interfaces or shared boundaries between components of the systems engineering environment. This figure shows the three defined interfaces: the User(s) interface with the Computer System Workspace, the User(s) interface with the external systems, and the Computer System Workspace interface with the external systems.

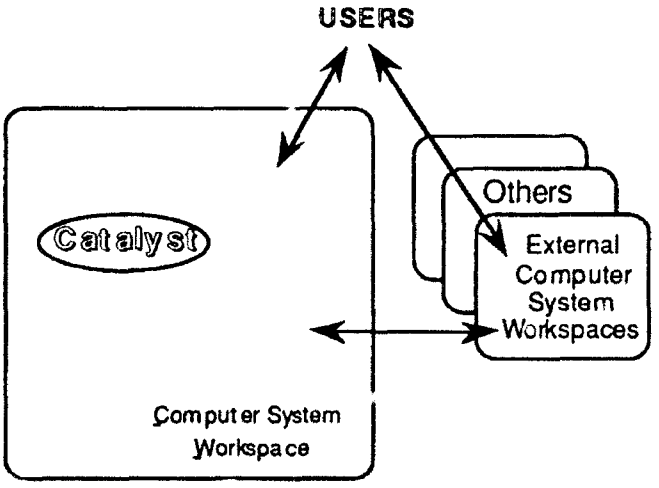


Figure 1.2-1. External View of the Automated Environment

As illustrated in Figure 1.2-1, the object named Computer System Workspace was viewed as a black box and was defined entirely in terms of its external operational behavior (i.e., in transitions from stimuli to response). This external model considers only those aspects that can be viewed from the outside. The external model has no knowledge of the internal state of the object. The external interfaces to the Computer System Workspace will determine the degree of portability to other platforms.

Figure 1.2-2 allows us to look at the systems engineering environment from a different point of view. Again, we have indicated the relevant interfaces with arrows. The figure shows Catalyst as contained within the Computer System Workspace, also referred as the Catalyst Environment. Clearly, many other arrows can be drawn to represent the existence of other interfaces, but our concern is only those interfaces that related to the Catalyst Environment. Catalyst has the following interfaces:

- User's host computer system, consisting of the computer hardware and the system software
- User's installed frameworks
- User's installed tools
- Users (employing the services of the user's host computer system hardware, system software and possibly installed frameworks
- External Systems (employing the services of the user's host computer system hardware, system software and possibly installed frameworks

Note the latter two interfaces were layered on top of the user's computer hardware, system software and frameworks. The program interfaces of the Catalyst software involve only the system software, user frameworks, and user tools.

As the complexity of the views grew, it became apparent that the interfaces must be scoped for the remainder of this survey. Therefore, for the focus of this effort, we are considering only these specified interfaces shown in Figure 1.2-2. These interfaces are relevant because they will be the ones specified and eventually built when Catalyst is implemented.

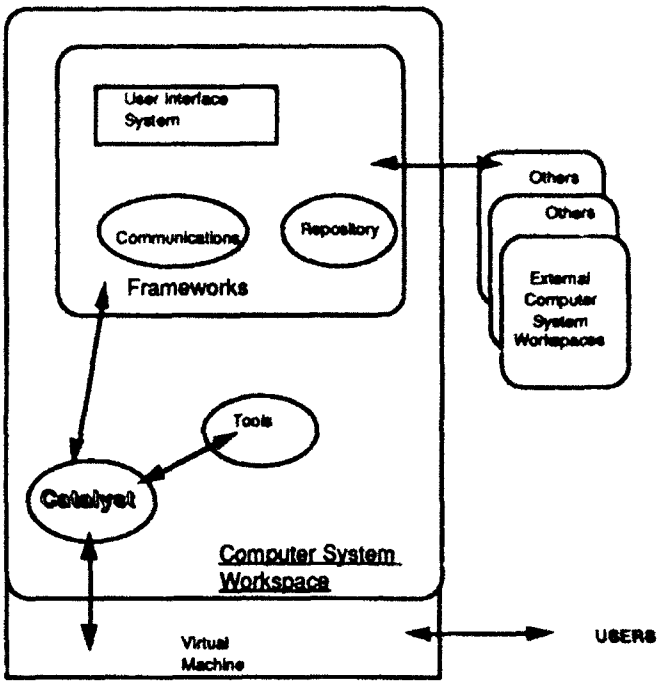


Figure 1.2-2. Internal View of the Catalyst Environment

1.3. Classification of Interfaces

The previous internal view of the Catalyst Environment helped to identify a classification of interfaces in the systems engineering environment. This classification scheme helped to focus our thinking, organize a large volume of information, and ultimately drove the study and analysis of interface standards. As can be seen in Figure 1.2-2 the arrows that point to Catalyst define three classes of program interfaces:

1. **Frameworks interface** which included several components that provided the following services:
 a. User interfaces allow the user(s) of the Computer System Workspace to communicate with the Catalyst Environment.
 b. Communications allow the Catalyst Environment to communicate with other external Catalyst Environments, target systems, file servers, and special database machines.
 c. Repositories allow storage and retrieval of project information by the Catalyst Environment.
2. The **Tools interface** allows tools in the Computer System Workspace to communicate with the Catalyst Environment. The Catalyst Environment, itself, was considered a toolset.

3. The **Virtual Machine interface** to the underlying hardware and system software supports the two interfaces defined above.

Catalyst also interfaces with the users and external systems; however, this interfacing was achieved through the services of the user's host computer system hardware, system software, and possibly installed frameworks. The program interfaces of the Catalyst software involve only the user frameworks, the user tools, and the virtual machine.

1.4. Interaction of Interface Classes

Each of the three interfaces defined in the previous paragraph is a subsystem or a collection of software that provides common facilities for the Computer System Workspace. Architecturally, the entire system included interface subsystems for each interface class. As illustrated in Figure 1.4-1, the Computer System Workspace encapsulates each of these subsystems in the systems engineering environment. This workspace was just one instance of an environment where the Catalyst Environment is used. Here, the subsystems or entities represent the interfaces.

Each interface subsystem depicted in Figure 1.4-1, actually consists of two interfaces:

- An inward interface to the Tools/Catalyst Environment
- An outward interface to the virtual machine or virtual machine interface

As seen in Figure 1.4-1, the inward interfaces consisted of user interfaces, communications, and repositories, all sharing boundaries or 'interfacing' with the Tool Interface. This inward interface determined the degree of interoperability between the Computer System Workspace and tools or other external workspaces. The outward interfaces consisted of the user interface, the communications, repositories, and the Tool Interface, all sharing boundaries or 'interfacing' with the Virtual Machine Interface. The Virtual Machine Interface shared its boundary or 'interfaces' with the virtual machine. This outward interface determined the Computer System Workspace's degree of portability across other virtual machines.

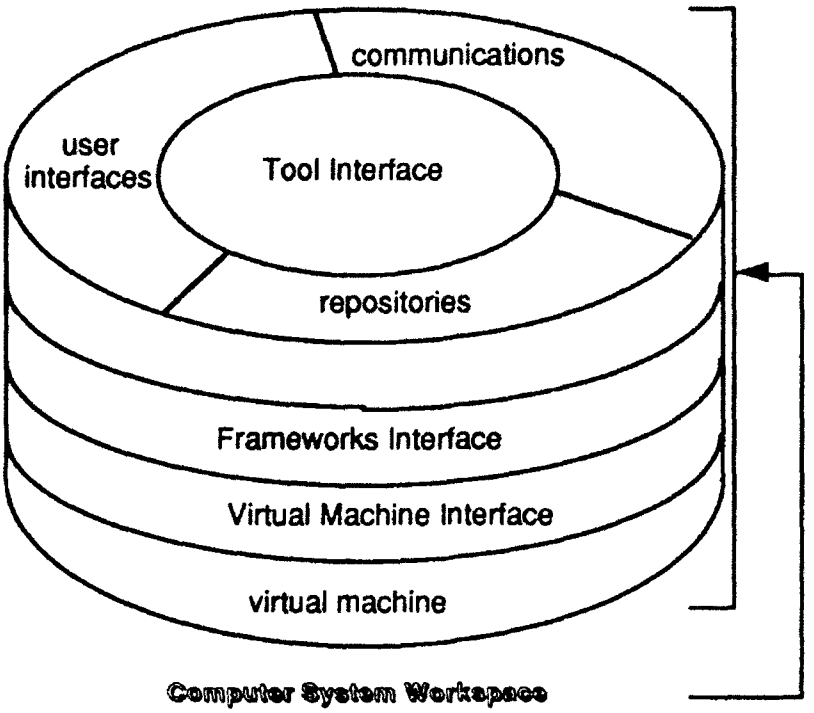


Figure 1.4-1. Relationship of Interfaces in the Catalyst Environment

Hence, we introduced the concept of an interface with two sides. Future investigations must therefore consider two interfaces, an inward and an outward, for each interface subsystem. This concept of a two-sided interface was a significant departure from the treatment of interfaces as single-sided filters or edges, and it better reflected the reality of implementation of each interface subsystem.

2. Interface Models

2.1. Model of the Frameworks Interface

A framework, itself, is a collection of software tools which provides common services shared among other software tools. Frameworks extend and enhance the services of the virtual machine, typically providing higher level, more powerful services for users and tools. Frameworks enhance a tool's integration and/or commonality through the shared use of common services. A framework usually provides services for user interfaces, communications, and repositories. Because each service has diverse operations, it was more appropriate to develop a model for each of the services rather than develop one model for the Frameworks Interface for Catalyst. Therefore, we

developed a model for user interfaces, a model for communications, and a model for repositories. The sections that follow discuss these models.

2.1.1. Model of User Interfaces

There was no standard definition for a user interface, but several of the following descriptions give the reader a general understanding of its dimension. For example, we can define a user interface as all aspects of system design that affect system use [SMI82]. More specifically, the MITRE Corporation narrowed the definition of a user interface to computer-based information systems, i.e., concern with those aspects of system design that influence a user's participation in information handling tasks [SMI86]. Using a different approach, Nash defined a user interface as a window system which draws on enhanced interaction facilities (bit-map display, mouse, menus) to shield both the user and the tool-writer from knowing the detailed aspects of the interaction protocol [NAS85]. On the other hand, Sisson referred to a user interface as 'human-computer dialog rules' so as not to be confused with the term 'interface'[SIS86].

In addition, Sisson developed a model for his human-computer dialog rules that can be applied to the definition of the Catalyst Environment User. Sisson extends the existing ISO (Open Systems Interconnect) model by adding three new layers, i.e., semantic, syntactic, and lexical as shown in Figure 2.1.1-1.

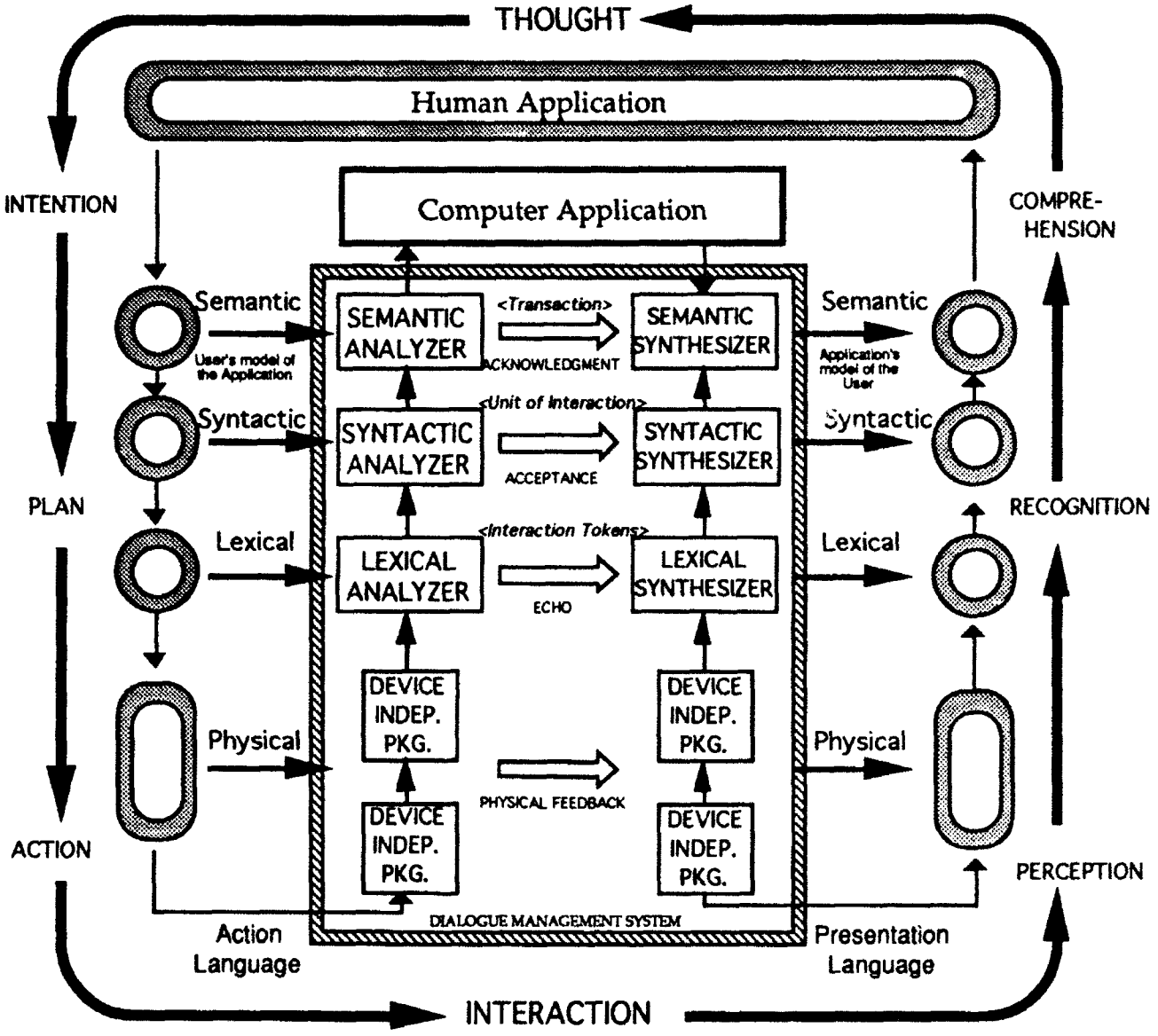


Figure 2.1.1-1. Layered Structure for Human-Computer Dialogue [SIS86]

The implementation of compilers and interpreters commonly use these layers. Tools, often referred to as compiler compilers (i.e., yacc and lex from the UNIX system), support these layers. Interfaces exist between the vertical arrows between adjacent layers; protocols exist between peer layers.

Human-computer dialog rules are the set of peer layer protocols along with the user feedback rules, and appear as horizontal arrows in Figure 2.1.1-1. Sisson suggested the members of SIGCHI (Special Interest Group On Computer and Human Interaction) standardize the set of human-computer dialog rules. Interfaces between the adjacent layers are implementation details and are standardized by implementors.

Sisson's model dictated that both the human and the computer handle the protocols. Because the human-computer dialog rules were designed and implemented by people who were interested only in computing, the resulting computer language must be learned by the human before a dialog occurs. However, the most desirable dialog rules have their foundation in natural language, that is, one in which the human learns the protocol and implements it on the computer.

The human and the computer must handle revision of the protocols. On the computer, a revision must be handled by replacing the software; for the human, the revision must be learned, and the possibility of confusion between the old protocol and the new protocol may occur. Sisson proposed that the model may provide an aid for minimizing confusion in such cases [SIS86].

Since implementing the modules of the standard OSI model usually results in finite state machine constructs and communicating sequential processes, Sisson expects implementations of his model will result in the same. This implied that a multi-tasking and/or multi-processor environment was required. Likewise, the language used to define the protocols should be compatible with the finite state machine implementation. If the language is compatible with lex and yacc, constructing a prototype of the system could help verify the utility of the standard and determine the coherence and completeness of the protocols.

2.1.2. Model of Communications

For communications, open systems and networking go hand in hand. Both the open systems movement and the users' desires were derived from a need to network disparate systems together and create a unified system. This

system must also be used and managed as if it were one logical system [JON90].

Networking and open systems are a long-term view of the computing world, and it assumes no technology-or-vendor-imposed barriers to accessing information. Barriers are still needed, but they are imposed by the users of the systems, not the systems themselves. A pure open system approach makes no assumptions about users' needs, but requires users to make no prior assumptions themselves by providing a fully integrated system.

The open network computing model is a model in which the physical location of processors, software, devices, and other resources is transparent and irrelevant to the user. The type of network and protocols at work would also be transparent to the actual user. The network becomes the primary source of information transfer. By concentrating a large part of the effort on eliminating barriers in this area, a foundation for openness will be established from which the rest of the computing environment can be built. In the future, entire systems will be built from the network infrastructure.

Communications subsystems are often designed to support point-to-point command and control communication. Where the propagation path includes the atmosphere or space, communication is, by nature, broadcast to all receivers within its transmission pattern. When communications are constrained to wires, cables or optical fibers, the communication facility is usually equipped with several alternative media. The wide-band communication media are shared by many users via a variety of multiplex schemes such as time-division, frequency division, and code-division, in various combinations [BEA90].

A communication path between two points is most often comprised of a number of serially connected links, with the ends of each link described as nodes. Some nodes may be merely terminal for transmission or receipt of information, while others may serve for switching the communications between different links, storing the messages for later transmission, converting the data to other forms, and encrypting it. [BEA90].

The fundamental system design considerations for a communications network are the medium and signalling methods used for transmitting each signal to be communicated. The nature of the information, its sources, and the character of the region through which transmission must take place is most important in determining these choices. For example, some situations require interactive two-way communications in real time, while others require only one-way transmission of previously prepared messages. With few exceptions, communication links in C-cubed systems make use of well-

known media and signalling methods to which international standards apply [BEA90].

As with most system design, the C-cubed systems design process may be characterized as building a complex system from carefully selected communications of available technology and subsystems. In this type of design process, one of the most valuable support elements would be up-to-date, easily accessible libraries of available components, subsystems, and existing interface definitions. An appropriate set of theoretical and heuristic relationships which govern the combination of system elements should be added to these elements. When a signal passes through a series of communications links, nodal and terminal devises the communication elements involved relate to the signal power, noise, and distortion. These relationships should be supplied in forms which make it easy to combine system components from the library and to estimate and compare technical characteristics of various combinations [BEA90].

Because most C-cubed systems handle numbers of simultaneous independent communications and data operations, a system estimation requirement was of special interest. The value for the system estimation requirement is the statistical estimate of time delays and system performance in the presence of large quantities of signal traffic. Although the individual queuing models on which this type of analysis is based is mathematically straightforward, non-random signalling patterns are more common in a military C-cubed environment than they are in civil telephone systems. This fact is true except in time of a civil emergency. During an emergency, most civil communication nets suffer saturation effects, at traffic levels beyond those for which they were designed. Military system must operate effectively under crisis (i.e., combat) conditions, which is the purpose they are intended to serve [BEA90].

The Catalyst Environment must have the capability to exchange data and information with other non-project-specific systems. Non-project-specific systems would include personnel management systems, financial management systems, hardware engineering systems, and other systems engineering environments. The 'other systems' with which a particular Catalyst Environment interfaces may vary with each program and installation. The Catalyst Environment must have the capability to generate and incorporate tools needed by a project to accomplish cooperating system communication.

As a first step toward international standardization of a communication interface, the International Standards Organization (ISO) developed an Open Systems Interconnection (OSI) Reference Model. The OSI model is a

framework for defining standards for linking heterogeneous computers and consists of seven layers:

- Layer 1 Physical
- Layer 2 Logical link
- Layer 3 Network
- Layer 4 Transport
- Layer 5 Session
- Layer 6 Presentation
- Layer 7 Application

Layer 1, the physical layer, is concerned with transmitting raw bits over a communication channel in the form of signals. The design issues here largely deal with mechanical, electrical, and procedural interfacing between networks.

Layer 2, the logical link layer, transforms a raw transmission facility into a line that appears free of transmission errors to the network layer. It accomplishes this data control task by breaking the input data into data frames, transmitting the frames sequentially, and processing the acknowledgement frames sent back by the receiver. Layers 1 and 2 are commonly implemented in industry standards such as Ethernet or Telnet.

Layer 3, the network layer, determines the chief characteristics of the interface message protocol-host interface, and how the packets (units of information) exchanged are routed. The design issues here are establishing the routing of the packets, ensuring that all packets are correctly received at their destinations, and in the proper order. The capability of the Internet Protocol part of TCP/IP is similar to that of the services offered by Layer 3. The capability of the Transmission Control Protocol of TCP/IP is similar to that of the services offered by Layer 4.

Layer 4 provides features of reliability, type, grade, and connection management.

Layer 5, the session layer, controls the dialog between two machines to establish and use a connection. The session layer is the user's interface into the network, by way of a dialog. Layer 5 is commonly implemented in ftp (file transfer protocol).

Layer 6, the presentation layer, performs functions that transform the network transmission. Formatting, text compression, and encryption are

services provided in Layer 6. The task of syntax checking of Layer 6 can be accomplished at the operating system level (i.e., UNIX).

Finally, Layer 7, the application layer, defines the specific messages and action taken upon the receipt of each message and is determined by the individual user. For example, E-mail may be provided in Layer 7.

Based upon the OSI Reference Model, the Federal Information Processing Standard adopted the Government Open Systems Interconnection Profile (GOSIP) to define a common set of data communication protocols or rules and conventions to allow conversation between each adjoining model layer. These protocols were developed by international standards organizations, primarily the ISO and the Consultative Committee on International Telephone and Telegraph (CCITT). GOSIP's common set of data communication protocols enable systems developed by different vendors to interoperate. This feature enable the users of different applications on these systems to exchange information. The details of GOSIP are further discussed in the supporting document for Interface Standards Studies.

Other research in the industry has continued since the standardization of the OSI Reference Model and a few do not support the model. For example, Cohen states that the seven layer OSI model is good for explanation, but it is no great revelation from an implementation standpoint [COH83]. He suggests that the layered model does not stand up well when trying to describe actual implementations. Cohen demonstrates his point by citing exceptions:

- The 7th layer is not necessarily the top.
- The 1st layer is not necessarily the bottom.
- Layers can mutually encapsulate each other.
- Layers can encapsulate themselves.
- There can be an infinite number of intermediate layers.

Nonetheless, the areas of communications and networking are mature and rapidly proliferating. There are many commercially available products in both hardware and software, and numerous standards have been adopted. The current trend is to treat the network as the system, rather than the system consisting of an isolated user.

2.1.3. Model of Repositories

The products stored in a repository can be varied; requirements and design specification, high and low level programs (load modules, assembly

programs, etc.), test input and output, documentation, and graphics. Associated with each item in the database may be a number of attributes such as the type of an item, stage of development, time of last update, access rights, author, version number, length of item, disk or memory address of item, etc. The interface to the database can be environment specific or it can be implemented by an available, possibly commercial, database management system file system , or library system [HOU87].

Historically, there are three database models; the functional, entity-relationship, and as the next generation, object-oriented. Database technology is mature in the use of relational approaches; there are may commercial products, and the Standard Query Language (SQL) is the most popular. However, the large-scale applicability of a relational database to CAD/CASE/CAE projects is questionable. Even though the E-R databases are the most popular, CASE applications bring a set of problems which, while not unique to CASE, are not fully solved by earlier modeling concepts. Consequently, object-oriented models for databases and repositories fit most appropriately for CAD/CASE/CAE applications.

Object-oriented databases are an emerging technology, and there are an emerging set of commercial object-oriented database (OODB) vendors. The OODBs are appealing because the data model more closely matches the real-world entities of a system, and the database language can be integrated with an object-oriented programming language. Object-oriented database technology was projected to be usable in the five year time frame.

The object-oriented databases have begun to grab commercial attention because of their advantages [LIV90]:

- Faster programming
- Easier database maintenance and extension
- Greater data integrity
- Databases more directly attuned to the way users think in the real world
- Efficient coupling of databases to distributed-processing networks
- Enhancement of team-based approaches to many tasks

Such databases are composed of software modules, called objects, in which data is combined with the information necessary to manipulate the data. The objects are arranged in classes that share common data and procedures. The schema of an OODB is built around the objects, classes, and other parameters. Most real-world information is in the form of objects. Objects are a more

natural, intuitive way of structuring information than tables in RDBMSs, especially when you are dealing with complex and multi-media-based data. OODB can be better suited to systems with heterogeneous, complex data involved in complex relationships and data types.

Some industry observers see OODBs as a natural match for such networks and the client-server architectures on which they rest. Objects, as self-contained, are modular pieces of information which lend themselves to being distributed among networks and servers; however, there is a price for such flexibility. An object directory is needed to identify machine objects, the users, and the users who are authorized to obtain objects. The use of networked OODBs can lead to potential problems unless the information is well-managed. This management could be handled by an object broker. The object broker could direct you to high-quality printers, a color printer, or a printer not being heavily used. This approach uses a system called a client-broker object architecture.

At a higher level of abstraction than the object-oriented database, a repository is really a database about system development. A repository is a database that defines and contains all the information relevant to the components manipulated by the workstation. A central repository contains the entire enterprise model storing meanings of diagrams rather than their physical representation. Cross-checking and correlating analysis/validation of plans, models, and designs are performed automatically, thus enforcing consistency [TER90].

To be successful, a repository model should [ST90a]:

- Provide a flexible way to organize information
- Allow an object to be viewed from more than one perspective
- Provide support for changing information within the repository
- Ensure the physical and semantic integrity of its objects (i.e., only complete objects, with necessary administrative information and documentation, can be stored in a repository)
- Enable adding site-specific policies for validating an object before it is entered in the repository
- Provide a means to examine the object within the repository

A repository model provides a logical way to organize and manage objects in the system, independent of the object's physical implementation. This eases the task of using underlying objects by not requiring the user to have knowledge of how they are physically stored. The logical grouping of objects

in the system has an advantage when physical objects must be relocated; the repository objects must be relocated; the repository merely updates the reference to the physical location of the object without changing the view presented to the user.

A CASE Repository can be further defined as the representation, in data, of all facts about the system under development, in a form which is independent of its mode of entry or subsequent analysis and reporting. To meet these requirements, the repository must be a no-loss representation of the system being described, that is, the repository must contain all the information conveyed by any of the notations used for development. It should be possible to regenerate any representatio· . given the proper transformation engine (i.e., CASE tool), from information contained in the repository [WEL89].

In addition, the facts in the repository should be kept in a non-redundant form so that changes to design objects and their relationships need only be made in one place, eliminating inconsistencies. For example, if a data element's name changes in an E-R diagram, it should automatically change in a corresponding data flow diagram. This supports an object-oriented model for a repository. Non-redundant storage also allows the contents of the repository to be more easily analyzed by CASE tools since the information is already broken down into its constituents [WEL89].

The repository constraints and checking mechanisms should ensure that information about the system being described does not violate the basic rules of a semantically correct description. No-loss representation and non-redundant storage make it easier to ensure the integrity (i.e., the accuracy and consistency) of design information place in the repository.

As computer systems grow in size and complexity and the volume of information to go into a repository increases, traditional magnetic tape storage is inadequate and optical storage becomes a viable solution. When the kind of information to go into a repository is examined, the traditional relational data models must be revisited. The relational model contains a hidden implicit assumption that all stored data will be either alphabetic or numeric. However, more than 25% of the contents of a repository will consist of images, graphics, or non-standard data types (possibly inc'uding voice) that tend to go beyond the relational database concept and will require object-oriented data models.

Comer developed a layered reference model for information storage that can be applied to the Repository Interface model for the Catalyst Environment [COM87b]. The layers in Table 2.1.3-1 are not strict layers, but have some overlap and reflect only general categorizations.

Table 2.1.3-1. Reference Model of Information Layers

Kind of Layer	Scope of Layer
Pragmatic [SOW84]	Meaning as related to current context and the user's expectations
Semantic	Meaning of the information, IRDS
Syntactic (external forms)	Products (Documents, Objects code) Programming Languages Specification Languages Report Formats Mail
Syntactic (internal forms)	Internal Languages Internal Formats (e.g., DIANA, Word Processing)
Objects	Ada data types and structures
Bitstream (Data link level)	Tape formats
Physical	Magnetic Tape Media Connectors - Mechanical Electrical

Each of the layers in the repository interface model has a different scope and addresses a different concern. The pragmatic layer is concerned with the cognitive process of the user applying the appropriate standards to the information. The semantic layer, which addresses the meaning of information, could use the standard of the Information Resource Dictionary System (IRDS) for cooperating system interfaces. The syntactic layer, concerned with both the external and internal forms of information, could use an applicable standard of syntax description format referenced in the IRDS. The bitstream layer, concerned with the data link level of information, uses the Network Protocol requirement to support Transmission Control Protocol/Internetwork Protocol (TCP/IP). As can be seen from the repository interface model, there is information flow at many levels of abstraction in the systems engineering environment.

2.2. Model of the Tool Interface

The model of a tool interface was driven by the definition of a tool. A tool can be defined as an instrument used in the performance of an operation. Tools provide specifically identified services by communicating with the user, interfacing with other tools, performing functions and retaining information. To provide these services, tools typically have four components as illustrated in Figure 2.2-1:

- A user interface
- A communications interface to other tools
- A processing component that provides the underlying capabilities and functionalities
- An information component for retaining information between invocations

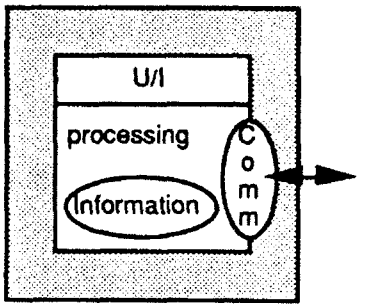


Figure 2.2-1. Model of a Tool

These four components of a tool are all layered and operate within a set of tools selected from different layers in the hierarchy of tools in the Computer System Workspace, commonly called a framework. Consequently, the interface for two tools was defined by the level of commonality between each of the tool's respective framework. The model for tool to tool interaction is shown in Figure 2.2-2. The tool communication with another tool was by way of its communications interface while both tools reside within their respective frameworks.

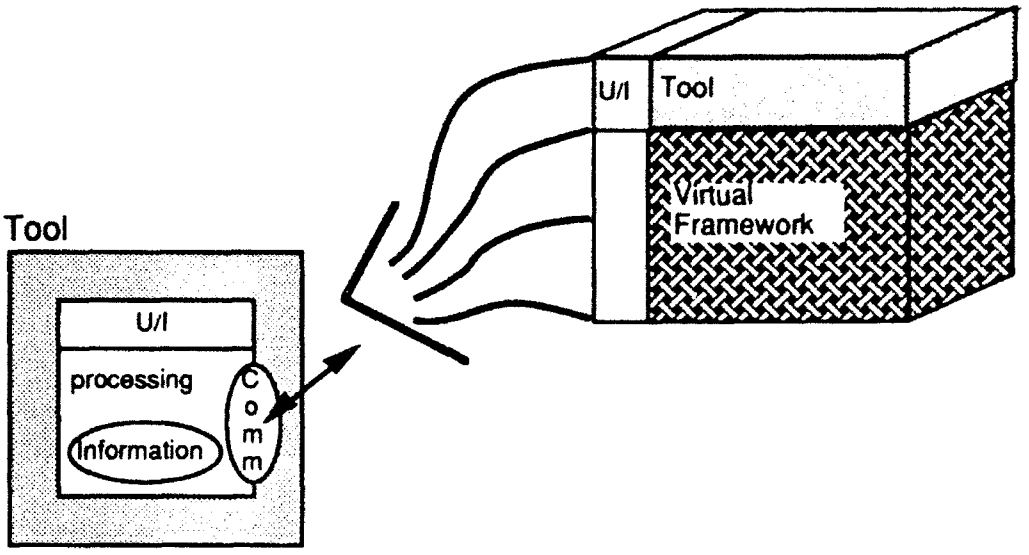


Figure 2.2-2. Model of Tool to Tool Interaction

The Tool Interface Model and its standards must support several kinds of design data: textual, tabular, line-graphic or photographic forms. These kinds of design data are all intended for visual use[2] [BEA90]. Textual data may include:

- Design studies
- Results of theoretical modeling or experimental tests
- Systems analyses
- Requirement documents
- Standards, including interface descriptions
- Reports on design activities
- Operations and maintenance documents
- Text Descriptions of subsystems

[2] Only very seldom does design data take the form of audio or video recordings, physical samples, or vials of scents, for human sensing by other than visual means. However, such items as physical samples of surface finishes may need to be referenced within the design environment.

- Message and signal formats to be handles in the C-cubed system

Any of this textual data may be supported by additional tabular and graphic information intended to accompany the text. While embedded tabular information can, and often will be formatted as text, in many cases it will be desirable to quickly determine aggregations, extract subsets, and create plots from the tabular data. Some word processors can display data from spreadsheets, but few spreadsheets can instantly accept data from a textual table, especially if the data is complicated by footnote references and graphic constructions such as surrounding lines or boxes. Thought should be given to whether standards dealing with text-tabular conversion will be important [BEA90].

Formatted text is substantially different to deal with than unformatted text even as the non-technical users of spreadsheets understand. Formatting improves readability of documents, even on a display screen, and there are a few widely used formatting standards among tools. In the approach taken to text formatting, the user should preferably not be troubled with the need to continually manage conversion of documents or graphics from one format to another. This suggests, for example, a document header which unambiguously defines any formatting instructions be provided [BEA90].

When dealing with graphics, some users will need to alter them, some will need to extract from then, and others should not even be allowed to change them. Because of the needs of an application (such as shading of solids), the complexities of graphic display devices and conversion software, and the need for compact storage of graphic data, many graphics-capable design programs have unique, often proprietary graphic formats [BEA90].

Users who must manipulate graphics will have no option but to use the programs supplied for that purpose by software vendors. In this case the most important missing element is a way to reference parts of the drawing, or locations in the two-dimensional area, or to reference indices known outside the drawing file. Some ability to add pointers and text annotation is desirable, as a limited form of interaction with graphics in any format; the pointer(s) and annotations, of course, can be in another format, electronically superimposed on the original graphic. Just as with text material, each graphic should carry identifying and context-defining information, along with source and revision data [BEA90]. All these factors must be considered in the model and the standards that apply to the process of information transfer and the information that is being transferred in the tool interface.

2.3. Model of the Virtual Machine Interface

In the previous discussion of the model of an environment, we defined the lowest level of a virtual machine as the hardware and the operating system. Before a model for the Virtual Machine Interface was defined, we established the concept of an operating system. An operating system can be defined as a collection of programs that coordinates the operation of computer hardware and software and provides the environment within which programs are executed. The specific services provided will, of course, differ from one operating system to another, but there are some common classes of services which can be identified. An operating system usually provides the functions depicted in Figure 2.3-1.

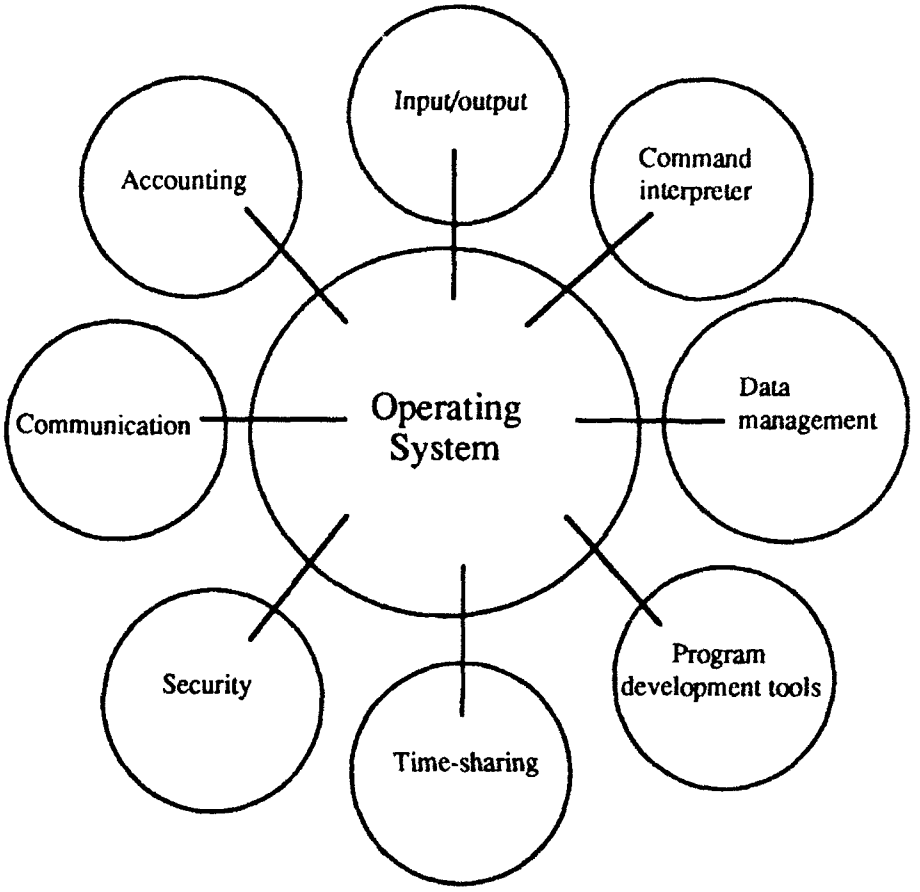


Figure 2.3-1. Operating System Functions [KOC84]

These illustrated functions can be described as follows:

- Input/output allows storage and retrieval of data, user interaction, and printing output on paper.
- Command interpreter reads the commands a user inputs and changes them into instructions the computer can understand.
- Data management allows the users to organize their data into files.
- Program development tools assist the user in writing and maintaining programs, includes compilers, assemblers, debuggers, and software maintenance systems.
- Time-sharing allows several users to execute programs on different terminals at the same time.
- Security protects one user from another and the operating system from the user.
- Communication is the ability of one computer to communicate with other computers and terminals to transfer programs and/or data.
- Accounting tracks the activities of the users for financial purposes.

These operating system functions listed above and the Catalyst system software and hardware formed the virtual machine that interfaces to users and tools. The relationship of the Virtual Machine Interface to other Catalyst interfaces was shown as part of the previous illustrations in Figure 1.2-2 and Figure 1.4-1. The virtual machine interface shares its boundary or 'interfaces' with the virtual machine. On the other side of the double-sided interface, the interface of the virtual machine shares its boundary with the user interface, the communication interface, the repository interface, and the tool interface by the flow of data, both outward and inward. The virtual machine was part of the virtual framework, and conceptually, interwoven with the framework as one entity. Consequently, the model of the virtual machine interface was determined by, and intimately associated with, the computer hardware and the computer operating system chosen for the Catalyst Environment.

3. Criteria for Evaluation of Interface Standards

In order to investigate and assess the potential use of current and emerging interface standards for the systems engineering environment, SPS established criteria to evaluate the interface standards for each class of interface. Some criteria were more applicable to one class of interface than another. The descriptions of the interface standards identified for each class of interface will included the following characteristics when appropriate:

- description

- type
- maturity level
- trend
- usefulness
- required hardware and software
- architecture
- generality
- integration mechanism
- uniqueness
- performance
- cost
- usability
- extensibility and tailorability
- maintainability
- portability
- administration support
- problems
- risk

Even through all these factors were important to the evaluation of an interface standard, the level of maturity was of primary importance for specification of concepts and state-of-the-art technologies which are implementable in the five-year time frame for development of the Catalyst Environment.

4. Evaluation of Interface Standards

4.1 User Interfaces

4.1.1 GUI

The world of Graphical User Interfaces (GUIs) seemed fairly simple in 1984, when Apple introduced the Macintosh. Back then, the genealogy was straightforward: Researchers at Xerox's Palo Alto Research Center (PARC)

begat the Xerox Star; Steve Jobs visited PARC, saw the Star, went back to Apple, and begat the Mac.

But even though there seem to be dozens of GUIs today, it's clear that they all still share similarities that reach below the surface of competition. Most GUIs have three major components [HAY89]:

- A windowing system
- An imaging model
- An application program interface (API)

In addition, these parts of GUIs have become industry standards [HAY89]:

- A pointing device, typically a mouse
- On-screen menus that can appear or disappear under pointing-device control
- Windows that graphically display what the computer is doing
- Icons that represent files, directories
- Dialog boxes, buttons, sliders, check boxes, and a plethora of other graphical widgets

Combinations of these three components appear in the descriptions of the user interfaces that follow.

4.1.2 The X Window System™

In March 1988, the Massachusetts Institute of Technology (MIT) released what may well become one of the most significant software technologies of the 1990s: Version 11 of the X Window System, commonly referred to as X11 or XWindows. X11 may not change the world, but it is likely to change the world of workstations. Vendors hope that X will lead to a software explosion similar to the one that occurred in response to the PC standard on microcomputers.

The uniqueness of X is, for the first time, portable applications can be written for an entire class of machines, rather than for a single manufacturer's equipment. Programmers can write in a single graphics language and expect their applications to work without significant modifications on dozens of different computers. Since X is a network-based windowing system, applications can run in a network of systems from different vendors.

The X Window System is the result of years of development work by researchers from both industry and the Massachusetts Institute of Technology

(MIT). Initially, X Windows was developed by MIT's Project Athena, funded by Digital Equipment Corporation and International Business Machines, along with contributions from many other companies. It was master minded by Robert Scheifler, Jim Gettys, and colleagues at MIT, though it owes some debt to the "W" windowing package developed by Paul Asente at Stanford. Although there have been numerous research versions of X Windows, Version 11 is a complete window programming package. It offers much more flexibility in the areas of support for display features, window manager styles, multiple screens. X11 provides better performance than previous X versions, and is fully extensible.

X Windows is described as a windowing system for bit-mapped, graphics displays that supports color as well as monochrome and gray-scale. Multiple screens can work together, with mouse movement allowed to cross physical screens. With X Windows, the screen layout or appearance and the style of user interaction with the system are left up to a separate program called the window manager. The window manager is just another program written with the X library, except that it is given special authority to control the layout of windows on the screen. X is somewhat unusual in that it does not mandate a particular type of window manager. Its developers have tried to make X, itself, as free as possible of a specified window management or user interface policy. And while the X11 distribution includes *uwm* as a sample window manager, individual manufacturers are expected to write their own window managers and user interface guidelines.

The X Window System is a network-oriented windowing system. An application need not be running on the same system that actually supports the display. As of September 1990, only TCP/IP and DECnet networks are supported, though the vendor promises that will change soon.

In the network system, an X-client must be prepared to respond to any one of many different asynchronous *events*. Events include user input (keypress, mouse click, or mouse movement) as well as interaction with other programs. This need to handle events is a major difference between programming under the X window system and traditional UNIX or PC programming. An X program does not use the standard C functions for getting characters, and the program does not poll for input. Instead, there are functions for receiving events, and then the program must branch according to the type of event and perform the appropriate response.

X Windows is extensible, that is, the code includes a defined mechanism for incorporating extensions, so that vendors are not forced to change the existing system in incompatible ways when adding features. These extensions are used just like the core Xlib routines and perform at the same level.

The X11 subroutine library provides a long list of functionalities. The X11 subroutine library (Xlib) is expected to be stable for several years, and to be, at least, a de facto industry standard. While there may be additions to this library, changes will not result in incompatibilities. Programs written with this library will not need major revisions due to software updates.

With X11 Release 2, control of X has passed from MIT to the X Consortium, an association of major computer manufacturers who plan to support the X standard. The Consortium was formed in January 1988, and includes virtually all large computer manufacturers. Many software houses and universities are associate members, who do not have a voice in controlling the standard, but receive advance access to newly released software. The MIT X Consortium has developed standards and conventions that stipulate requirements for other tools to interface with the X Window System.

X-Windows has become a de facto industry standard and network protocol for windowing and graphics. X-Windows has been accepted in the computer windowing and graphics industry for three reasons. First, it provides a high performance network protocol for windowing and graphics. Second, it is independent of workstation hardware and operating systems. Third, it provides these capabilities in a network-transparent manner. These features imply that with the appropriate communications protocols, an application that uses the X-Windows interface can be displayed on any workstation or personal computer within the network [RUD89].

X-Windows has been adopted by the three organizations that are specifying open software environments: X/Open, OSF, The National Bureau of Standards and Technology, and the IEEE 1003.0 Group. X Windows provides the user interface for the NIST Application Portability Profile (APP) through remote graphics protocols by using the specification of ANSI-STD-X3H3.6. FIPS 158 is an APP User Interface Component for X Windows. The X Window System is being adopted as a standard by nearly every workstation manufacturer, and should eventually replace or be supported under their proprietary windowing systems. Versions will also be available for personal computers in the near future.

4.1.3 OSF/Motif™

As part of a movement to establish standards for user interfaces, the Open Software Foundation (OSF), in 1988, asked major software developers to submit graphic user interface technologies for consideration as part of a standard operating environment for UNIX. To most people's surprise, the OSF chose pieces from three companies - DEC, Hewlett-Packard, and

Microsoft. OSF's product, Motif, looks like Microsoft's Presentation Manager (PM), uses parts of the DEC and Hewlett-Packard application program interface (as well as the three-dimensional windows from Hewlett-Packard's NewWave), and is based on the X Window System. OSF/Motif, OSF's first offering, is a graphical user interface combining the following elements:

- A toolkit
- A presentation description language
- A window manager
- A style guide

The **toolkit** is a rich and varied collection of widgets (predesigned window elements) and gadgets for building OSF/Motif applications. The toolkit provides a standard graphical interface upon which the window manager is based. The behavior of the toolkit conforms to Microsoft's Presentation Manager (PM), ensuring an easy transition between DOS and workstation environments. Toolkit widgets provide a 3-D reference appearance that gives users real-world, visual cues to the effects of their actions.

The **presentation description language**, called the User Interface Language (UIL), is supplied by OSF/Motif and allows application developers and interface designers to create simple text files which describe the visual properties and initial states of interface components. Changes to components are made in the text file, eliminating the need to change application code when tuning an interface.

The **window manager** works with the toolkit to manage the operation of windows on the screen. The window manager provides functions for moving and resizing window, reducing windows to icons, restoring windows from icons, and arranging windows on the workspace. The OSF/Motif window manager provides compatibility with PM behavior. An additional OSF/Motif window manager feature is the icon box. The icon box contains icons for all windows operating under the window manager.

The **style guide** describes the standard for the window manager and the toolkit behavior. The style guide provides usage, providing application writers with guidelines for using toolkit widgets, widget writers with guidelines for designing new widgets, and window manager writers with guidelines for designing new or customized window managers. Together, these four elements, the toolkit, the presentation description language, the window manager, and the style guide, provide a standard of user interface behavior for applications.

OSF/Motif runs on the workstations of Hewlett-Packard, Digital Equipment Corporation, Sun Workstation, and Interactive. Like most software products in their early stages, some bugs exist, and are being uncovered by users. OSF/Motif is currently in Version 1.0; Version 1.0.3 was released with some fixes to bugs, with Version 1.1 soon to be distributed. Current documentation explains the widgets in detail, but could be more informative about how these pieces fit together for an application according to some software developers. Both these areas are being addressed in the plans for the next version through feedback from networked users. This presents a degree of risk; however, because Motif is backed by standardizing organizations, this risk is controlled.

Motif is fast becoming an industry de facto standard. Following the announcement of Motif, many companies announced support for the OSF standard and began tweaking their graphic user interface software to be compatible with it. For example, as of April 1990, over 600 companies have licensed OSF/Motif source code, representing 25 countries and 80% of worldwide computer suppliers. Endorsements include The European Economic Community, '88 Open Consortium, General Motors, American Airlines, and the Marriott Corporation. OSF/Motif is more popular than Open Look, with 73% of software vendors working in UNIX having elected to support OSF/Motif. Presently, Motif runs on more than 100 hardware platforms and 38 operating systems [WAG90].

OSF/Motif won five major industry awards in 1989:

- *Byte Magazine*, "The Byte Awards"
- *VARBUSINESS*, "Top Products of the Year"
- *UNIX Today*, "The Top Ten Unix Stories of 1989"
- *UNIX World*, "Best Products of 1989"
- *International Design Magazine*, "Design Awards"

The U.S. Government's response to OSF/Motif has been very positive as indicated by the acceptance by the U.S. Air Force and U.S. Navy, and OSF/Motif's references in U.S. Government RFPs.

4.1.4 Open Look™

Open Look, designed by Sun Microsystems, Inc. for AT&T in 1988, is based on a technology licensed from Xerox. Open Look is the Graphical User Interface for Sun's XView, a retarget of SunView to run on X11. XView is an object-oriented toolkit that runs on a server-based window system, whereas SunView is a existing kernel-based window system. The Open Look GUI is

superior to SunView's user interface because it offers programmers greatly enhanced functionality and permits visual consistency with other Open Look GUI applications [JAC89].

A goal of Open Look is to provide consistency between applications so that users can easily switch between applications. For example, throughout the system and across applications, a given mouse button is used for only one function. Open Look is also designed to be independent of the hardware and software that it runs on and to accommodate different keyboards, mice, and screen resolutions. Because Open Look provides the same interface across over twenty platforms, users need to learn the application only once. Open Look supports access to a large range of network resources [RUD89].

Open Look is the most popular front end to UNIX and is included as the standard interface of AT&T's UNIX System V.4., at no extra charge. Open Look makes UNIX easier to use by eliminating the complicated UNIX commands. The user interface uses push-pin icons, and 3-D elements with drag and drop features provide the user interface with an intuitive way to move files around the desktop.

Open Look is part of Open Windows, a complete development environment, and is a standard at a higher level than X Windows. Both, Open Look and Open Windows, together, support the user with tools to quickly create applications, with ready-made features, like easy to use DeskSet graphical productivity tools. In a manner similar to the Macintosh toolbox, Open Look provides an extensive set of higher level user interface toolkit routines for the application developer [RUD89].

Open Look has been very successful with independent software vendors, in-house developers, and end users. Over 300 applications are in development today, by companies such as Lotus, INFORMIX, Island Graphics, Interleaf, and Frame. Open Look has a degree of stability since it has the full support of a company that leads the workstation industry in worldwide shipment.

Even though Open Look is a good approach to user interface integration, several issues still remain to be resolved; the lack of uniformity across tools, the standards being at too low a level, and performance, complexity, and human factors [RUD89].

4.1.5 GKS

The Graphical Kernel System (GKS) is a set of basic functions for computer graphics programming, usable by many graphics producing applications.

These functions are taken as a whole and are called the Graphical Kernel System (GKS) [AME84]:

- Outputting graphical primitives
- Controlling the appearance of graphical primitives with attributes
- Controlling graphical workstations
- Controlling transformations and coordinating systems
- Generating and controlling groups of primitives called segments
- Obtaining graphical input
- Interpreting groups of device-independent instructions called metafiles
- Inquiring the capabilities and states of the graphics system
- Handling errors

These functions define an application level programming interface to a graphics system and provide the following features [AME84]:

- Allows graphics application programs to be easily transported between installations
- Aids graphics applications programmers in understanding and using graphics methods
- Guides device manufacturers on useful graphics capabilities

Much of the early design methodology for GKS was developed during the Workshop on Graphics Standards Methodology held in May, 1976 in Seillac, France under IFIP WG5.2 sponsorship. GKS itself was originally developed by the West German Standardization Institute in 1978 and was subsequently refined extensively during the period from 1980-1982 by Working Group 2 of the Subcommittee on Programming Languages of the Technical Committee on Information Processing of the International Standards Organization (ISO TC97/SC5/WG2). The resulting draft, International Standard ISO/DIS 7942 (ISO GKS) served as the basis for the American National Standard GKS (ANS GKS). GKS was adopted as a Federal Information Processing Standard (FIPS) in April, 1986 [SKA86]. The function of business graphics is supplied through GKS utilizing the specification of the ISO 7942 and ISO 8651K for the NIST Application Portability Profile (APP). The GKS ANSI X3.124.1 is identified as a standard in CALS.

ANS GKS was heavily influenced throughout its development by the work of the Graphic Standards Planning Committee of the Special Interest Group on Computer Graphics of the Association for Computing Machinery (ACM

SIGGRAPH GSPC). This work, know as the Core System Proposal, was published and widely distributed in 1977 and revised in 1979.

All the functional capabilities of ISO GKS are found in the ANS GKS. In addition, the ANS GKS contains the following:

- A new minimal output level
- Bindings of GKS functions to actual programming languages (i.e., FORTRAN, Ada)
- Less restrictive definitions of a conforming program and a conforming implementation
- Data records for input parameters

GKS is described in abstract terms, in order that it may be useful to applications in a wide range of environments with differing programming languages and communication protocols. A number of details of GKS are deliberately not specified so as to provide the freedom to adapt implementations to different environments and different requirements. Before GKS can be used by an application program programming with its specific language (host language), two further stages of specification are required [AME84]:

1) Language binding
2) Implementation

A language binding must instantiate abstract functions and data types of GKS in terms of the constructs available in the host language. GKS provides guidelines that should be observed when binding GKS to a host language. The implementation set of language specific facilities must then be provided using the facilities of a particular machine and operating system. One form of GKS implementation is a module or library of modules written for a specific programming language and conforming to a GKS language binding. GKS also provides guidelines for implementing GKS, with allowable differences both global and workstation dependent differences.

Graphics is an area where the slow standards process has been unable to keep pace with the technology. Unlike programming languages and syntax as a continuing focus of standardization activity, there has been a long period of graphics history where regional de facto standards have prevailed and slowly international standards have evolved.

The GKS standard has not been revisited since 1982, and advances in computer graphics capabilities during that period have created a need for

enhanced standards that include additional primitives and attributes to handle those new capabilities. Revision of GKS is focusing on a new application programming interface (API) projected to be released in 1993. The revision will be fairly conservative, and will correct known errors and consider adding a few capabilities such as he fill-area set primitive from GKS-3D and an extended set of line and marker types [CHI88].

GKS is being utilized by national and international communities who are now proposing modular graphics standards based on singular functionality. For example, GKS is proposed as a standard to draw machine parts using very high level graphics concepts called primitives.

One of the problems with GKS is that different implementations of the standard do not always produce identical output from the same program. Many new graphics packages are emerging as de facto standards, and many implementations of standards are built on top of other packages, or even other standards, leading to poor efficiency and slow performance [JER87]. Even so, the GKS is utilized in the Project Athena, a Massachusetts Institute of Technology campus wide, high-quality computing system based on a large number of networked workstations to support 2-D drawing packages [CHA90].

4.1.6 Microsoft Windows

Microsoft Windows is one of the many packages for the MS-DOS and Xenix operating systems in a large product line offered by Microsoft Corporation. Microsoft Windows is an extension of the DOS operating system which allows the user to integrate the different tasks he performs on his PC and thereby increase his efficiency. Microsoft Windows is an advanced new operating environment that bridges the gap between today's most popular and tomorrow's most powerful applications. Microsoft Windows allows the user to work with multiple applications and quickly switch between them without having to quit and restart each one. Moreover, the user can transfer information from one program and use it in another. Microsoft gives the user a visual way of working by organizing his work in windows, rectangular areas on the screen in which he uses his applications. When the user has a hard disk or a memory expansion card, he can swap programs and run more programs than normally fit in memory at the same time [STA87].

A dozen programs are included in the Microsoft Windows package, including a calendar, print spooler, clock, notepad, calculator, electronic cardfile, a terminal program, and an MS-DOS Executive file manager for easy access to DOS commands. Microsoft Windows has a option with an easy-to-use word processor, Write, and a drawing program, Paint. The applications have drop-down menus, icons, dialog boxes, and the ability to display multiple windows.

Microsoft Windows also has a Development Kit that gives the user the tools to develop sophisticated, portable graphics applications that use the popular Windows features such as drop-down menus, and dialog boxes. The Kit includes utilities, sample code, dialog editor, font editor, debugger, windows, and libraries all with documentation. Programs developed with the Microsoft Windows Software Development Kit are portable to any microcomputer running the Microsoft Window operating environment. The user can take advantage of the graphics devices and printer without the need to write drivers.

The minimum system requirements are an IBM PC or 100% compatible, 320K memory, DOS 2.0 or higher, two double sided disk drives, and a graphics adapter card. The recommended memory and disk storage is 512K memory, and one double-sided disk drive and a hard disk. A mouse is optional, but to obtain a color display, the user will need a personal computer equipped with the IBM Enhanced Graphics Adapter or compatible card. Color is not generated with an IBM/Color/Graphics Monitor Adapter or compatible graphics adapter card [STA86].

4.2 Communications

4.2.1 GOSIP

Related to the OSI model, the Government Open Systems Interconnection Profile (GOSIP) was established by the Federal Information Processing Standards (FIPS) 146. GOSIP is a suite of OSI Standards developed by a National Bureau of Standards (NBS) which will define an implementation profile for certain government contracts. NBS is using application profiles developed from the General Motors' Manufacturing Automation Protocol/Boeing's Total Office Protocol (MAP/TOP) organization as a baseline for its suite. All federal agencies must specify OSI-compliant products after August 1990, or are required to obtain a waiver. Gateways will be required to communicate between TCP/IP and OSI.

GOSIP defines a common set of data communication protocols which enable systems developed by different vendors to interoperate and enable the users of different applications on these systems to exchange information. These Open Systems Interconnection (OSI) protocols were developed by international standards organizations, primarily the International Organization for Standardization (ISO) and the Consultative Committee on International Telephone and Telegraph (CCITT). GOSIP is based on agreements reached by vendors and users of computer networks participating

in the National Bureau of Standards (NBS) Workshop for Implementors on Open Systems Interconnection.

GOSIP uses the seven-layer Open Systems Interconnection Model (OSI model) to define its specific communication architecture in the standard itself [USA89]. This architecture provides precise definitions of the functionality at the interface to each layer. Each higher layer hides details managed by lower layers, thus providing a suitable abstraction for communication at that level. Consequently, the OSI is a framework for identifying standards for linking heterogeneous computers by the GOSIP standard. Currently, GOSIP supports the Message Handling Systems and File Transfer, and Access and Management applications. GOSIP also supports interconnection of the following network technologies:

- CCITT Recommendation X.25
- Carrier Sense Multiple Access with Collision Detection (IEEE 802.3)
- Token Bus (IEEE 802.4)
- Token Ring (IEEE 802.5)

Additional applications and network technologies will be added to later versions of the GOSIP standard.

Boeing's Total Office Protocol (TOP) and General Motor's Manufacturing Automation Protocol (MAP) are industry products designed to be consistent with the OSI model and the GOSIP standards. MAP is used in the design of the Computer Aided Avionics Project Environment (CAAPE) by McDonnell Douglas Corporation, Douglas Aircraft Company. CAAPE is an integrated set of methods, tools, and procedures that are needed to develop and support avionics/electronics subsystem integration and acquisition programs through the entire life cycle, in accordance with customer requirements. CAAPE is being developed to share and incorporate information and resources with non-CAAPE networked systems such as computer aided software and hardware development environments. Because the implementations of the OSI differ, network-related conflicts may occur as these implementations of the OSI emerge [NAS86].

As GOSIP and OSI comes into widespread use, there will be an increasing migration to efficient hardware/firmware implementation of the OSI protocol stacks. This migration will provide highly reliable and efficient inter-computer communications [RUD89].

4.2.2 TCP/IP

A series of DoD military standards have been developed to allow communication between dissimilar computer hosts supplied by different computer vendors. These standards include Internet Protocol (IP) MIL-STD-1777, Transmission Control Protocol (TCP) MIL-STD-1778, File Transfer Protocol (FTP) MIL-STD-1780, Simple Mail Transfer Protocol (SMTP) MIL-STD-1781, and Telnet Protocol MIL-STD-1782.

The Transmission Control Protocol/Internetwork Protocol (TCP/IP) is the most popular of these standards. TCP/IP protocol has been used by the DoD-based ARPANET since the mid '70's to link a variety of computers among government, academic, and industrial laboratories. The protocol eventually migrated to Berkeley, where it was adopted for UNIX implementation as the Internet protocol. It has been widely distributed since that time, and is popular because it offers a high degree of interoperability among applications running machines that support different architectures and operating systems [DAY89]. TCP/IP commands allow the user to do the following:

- Transfer files between computers interactively
- Log in to remote computers and start a shell interactively
- Execute commands on remote computers interactively
- Send mail between users interactively

TCP/IP resides on top of the physical layer of local area networks (LAN's) bus protocols such as Ethernet and also on long haul networks based on x.25 and other physical protocols. TCP resides at Layer 3 (Network) of the OSI Reference Model to provide the chief characteristics of the interface message protocol-host interface, and determines how the packets (units of information) exchanged are routed. IP resides at Layer 4 (Transport) to provide features of reliability, type, grade, and connection management. Both are independent of the underlying physical layer. To use TCP/IP, the user must have the appropriate hardware:

- An Ethernet cable
- A drop cable from the host computer to a transceiver
- An Ethernet board

The Ethernet cable is the physical foundation of the network; it supports several layers of software. The first layer, Internet Protocol (IP), supports the Transmission Control Protocol (TCP). The TCP creates a virtual circuit, or a data path in which data blocks are guaranteed to be delivered to the target

machine, and in the correct order. Messages are sent from the sender to the receiver until the receiver sends back a message saying that all the data blocks have been received in the correct order. The TCP layer enables and supports applications such as TCP/IP commands.

TCP/IP is a mature standard, and extensive documentation and user support is available in the networked software communities. This communications protocol is a procedure with a well defined format that allows two or more systems to communicate across a physical link. TCP/IP provides a fast, standard, and reliable means of communicating with other systems. The DoD is committed to migrate from DoD protocol standards to international standards. This process will be slow because of the large installed computer and software base within the DoD [RUD89].

4.2.3 NFS

Network File System (NFS) is Sun Microsystems answer for file system integration by providing transparency in location of files. The NFS is typically the networking component in a user constructed framework utilizing UNIX, and Open Look. This framework is the underlying infrastructure that manages information, computing resources, inter-tool communication, tool execution, user access, and input/output for all user interactions with a computer based collection of tools. Tools that were designed to utilize the local file systems can run without change and access files located on different computers within the network, provided that the network is using NFS [RUD89].

NFS provides the user the following advantages:

- Transparent access to shared data
- Improved data consistency among network members
- Simplified systems administrative tasks
- Reduced local disk space
- Easier coordination of teams
- Transparent network topology
- Easier migration from time-sharing to networked environments
- Open, vendor-independent networking architecture

NFS has been extended to the PC world by the product PC-NFS. PC-NFS features provide the following:

- A common file system and simplified system administration for PCs and other client NFS systems
- Exchange of information with other client NFS network users
- Access to files elsewhere on the NFS network
- Storage and backup facilities of PC and client NFS files on the NFS network
- Access to other UNIX applications via remote login
- Spooling of PC and NFS files for printing on NFS network printers
- Clustering of PC and NFS clients with a proven LAN

The users of a PC-NFS receive the following benefits:

- Sun workstations, other client NFS systems, and PCs are under one network
- Standard PC applications can co-exist in the Sun/NFS environment
- PC users have intelligent engineering terminals, and have a window to the UNIX world and applications
- Sun or other NFS servers as LAN servers and PCs all share resources
- Terminal emulation for easy access using Telnet protocols over Ethernet to other and multiple UNIX systems
- More storage space over the network and greater performance with floppy based PCs

Issues that file system integration still needs to address are the multiplicity of the file systems, and the data structures and the meaning of the data being embedded in tools. Advances in the related area of database technology could help to solve these file system integration problems [RUD89].

4.3 Repositories

4.3.1 Standardization Groups

4.3.1.1 OMG

The Object Management Group, Inc. (OMG) is a high powered computer-industry trade association promoting object-oriented technology to integrate multi-vendor applications across heterogeneous networks. Some members of OMG are AT&T Co., Data General Corp. Hewlett-Packard Co., Sun

Microsystems, Philips Telecommunications, Objectworld, Inc., Mentor Graphics, and Object Design, Inc. with the support of ANSI.

In May 1990, the Technical Committee (TC) of the OMG developed a preliminary draft of a the OMG Standards Manual which contained an Abstract Object Model and a Reference Model as the foundation for object-oriented development. The OMG Object Model provides an organized presentation of the OMG object concepts and terminology. The Abstract Object Model partially defines the model of computation seen by OMG compliant applications, and by end users. Its purpose is to define a conceptual framework for OMG proposed technologies, and in particular, to motivate the basic design choices to be made in proposing and adopting specific technologies. One goal of this Abstract Object Model is to avoid placing unnecessary restrictions on the possible models of the proposed technologies. The requirements embodied in the OMG Abstract Object Model are those considered "essential" to the concept of object technology [SOL90].

In the Abstract Object Model, an interface is defined as a description of a set of possible uses of an object. Specifically, in interface describes a set of potential requests in which an object can meaningfully participate. An object satisfies an interface if it is meaningful in each potential request described by the interface. An interface can both describe what a particular object can do, as well as describe how a client intends to use an object [SOL90].

However, the OMG Abstract Object Model does not define the components of an OMG system or their interfaces, nor the structure of OMG applications. These issues are addressed by the OMG Reference Model [SOL90].

The OMG Reference Model for the Object Management Architecture (OMA) forms a conceptual road map for assembling technology that satisfies OMG's Technical Requirements. The Reference Model is intended for three audiences:

1) The OMG itself. The Reference Model provides a framework for guiding the process of soliciting and evaluating distributed object management technology.
2) Potential technology provider. The Reference Model provides an architectural structure for positioning and presenting proposed technology in relation to others.
3) Application developers and the software industry, in general. The Reference Model articulates OMG's vision for highly interoperable applications and services using object technology.

While the structure of the Reference Model will influence the high level architectural and component designs of specific proposed approaches, it accommodates a variety of different design solutions. The function of the Reference Model is more to map out areas to be addressed than to impose design constraints except at the highest, architectural level. It puts in place only sufficient structure to allow requirements to be defined and solutions to be proposed [SOL90].

The OMG Reference Model identifies and characterizes the components, interfaces, and protocols that comprise OMG's Object Management Architecture, but does not itself define them in detail. Detailed interface and protocol specifications will be evaluated and accepted by OMG through the process of technology sponsorship. Specifically, the Reference Model addresses:

- How objects make and receive requests and responses
- The basic operations that must be provided for every object, whether through classes, instances, or the OMA infrastructure itself
- Object interfaces that provide common facilities useful in many applications

The OMA defines an interface that allows a standard way to issue requests to conforming objects and to receive responses. In addition, the OMA identifies methods that all classes must support and optional facilities with class interfaces that are useful to a wide range of applications. The OMA has four major components:

- The Object Request Broker
- The Object Services
- The Common Facilities
- The Application Objects

The Common Facilities provide a uniform semantics that is shared across applications making OMA compliant application systems easier to use. The kinds of facilities that are candidates for Common Facilities are cataloging and browsing of classes and objects, link management, reusable user interfaces, the help facility, electronic mail, common access to remote information repositories, interfaces to external systems, and object querying facilities [SOL90].

The Application Objects have the ability to better wed single-minded functionality with other application classes. CASE tools are expected to fall within the Application Objects classification [SOL90].

Once a standard for Object Management Architecture is firmly in place, software developers and end-users would not have be concerned about the compatibility of specific programs as they evolve, as long as the programs follow standardized object design.

4.3.1.2 OODBTG

The Object-Oriented Databases Task Group (OODBTG) of the Database Systems Study Group (DBSSG) is currently developing a reference model, definitions of user roles, and interfaces for Object Data Management (ODM). The DBSSG is one of the advisory groups to the Accredited Standards Committee X3 (ASC/X3), Standards Planning And Requirements Committee (SPARC), operating under the procedures of the American National Standards Institute (ANSI). The OODBTG was established in January 1989, with the objective of determining which, if any, aspects of object databases are, at present, suitable candidates for the development of standards. In addition to stating informal definitions and requirements for an (ODM) and defining a common reference model for an Object Database, the OODBTG will make recommendations regarding future ASC/X3 standards activities in the areas of data management, data languages such as SQL, IRDS development, and related standards. Workshops for potential ODM standardization are planned for 1990 [OTI90].

The typical interfaces and users' roles which are envisioned for systems within the scope of the ODM reference model encompasses one or more user interfaces, in addition to the application program interfaces. The configuration of the connection of application program interfaces to the ODM system is characterized in terms of:

- Language characteristics of application program and/or the data language of the ODM system
- Level of automation of persistence
- Number of object spaces
- Number of operation execution spaces

Another dimension for characterizing an application program interface to an ODM system is whether structural access is allowed. Structural access involves simple, low-level messages to the database where the arguments

and results of operations are very closely related to the state or information encapsulated inside of the receiver object. In the limit, structural access may reduce to navigational fetch and store operation which violate encapsulation [OTI90].

As of October 1990, the Reference Model for Object Data Management is in its revised draft form. As these efforts continue, standardization of object databases and their interfaces will become more technically mature and stable.

4.3.2 Existing Standards and Standard Work in Progress

4.3.2.1 ATIS (Atherton Tool Integration Services)

ATIS (Atherton Tool Integration Services) is the draft, as of September 1990, of the ANSI work towards the next version of IRDS standard, as defined by WG3 document number ISO/IEC JTC!/SC21 WG3 N1020. The ATIS Reference Model, as shown in Figure 4.3.2.1-1, specifies four layers of service: the object, version, configuration management, and work flow layers, all based on the underlying data modeling approach that supports representation, storage, and retrieval of the information resources of an enterprise. The division is somewhat arbitrary, as any such division must be, but provides a framework for understanding the features of an information system, and provides a classification scheme for comparing the features provided by different systems [BEY90].

Each layer depends on the lower layers for services and provides additional services to the layers above. So the version management layer depends on the object layer for the representation of objects and associations between them; the configuration management layer depends on the versioning layer for the representation and manipulation of multiple versions of a configuration; and the work flow layer depends on configuration management to handle entire configurations as single objects [BEY90].

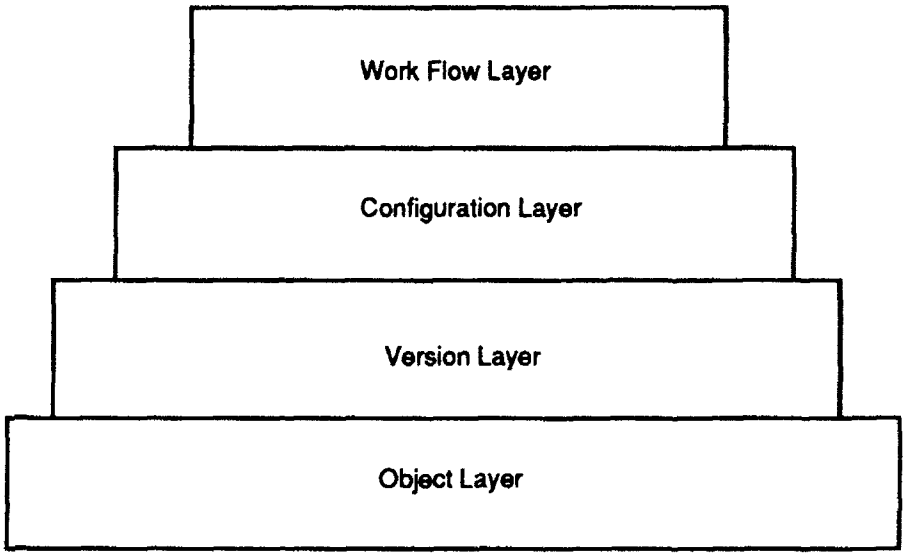


Figure 4.3.2.1-1 ATIS Layers [BEY90]

The layers of the ATIS architecture do not prevent access to the features of lower layers directly. This lessens the usefulness of the layers themselves as software constructs, but makes the system built more efficient, since necessary features of each layer do not need to be specified to make them available at the next layer.

ATIS is an object-oriented approach to the integration of tools that provides a set of interfaces that support schema-driven dispatching of behavior. ATIS is presented in the form of a type hierarchy that specifies abstract data types, their properties, and their methods. Each element (called an object) is an instance of a type (called a class). The types are related to each other with a hierarchy that defines the inheritance model for each chosen view of the world. The state of an element is indicated by the values of its properties (called attributes). Each class has properties that are given values when an element of that type is instantiated. An object's behavior is defined in its methods (similar to a procedure or function without inheritance).

Objects have unique identity with the system and are manipulated by messages. A message is sent to an object, and each object understands a limited set of messages. Each message is mapped to a method on the object.

The behavior of an object is visible only to the object itself. The following messages are among those defined in ATIS:

1) Get-prop – reads the properties associated with an element
2) New – defines a new entity or relationship
3) Set-prop – changes the values of properties associated with an object
4) Free – removes an object from the system
5) Rename – changes the name of an object

This paradigm can be extended by two or more elements sharing some properties or behavior can be defined as a single abstract type containing this shared behavior. The original types will then define this new type as their superclass.

Each new type inherits the properties and methods of its superclass. When a type inherits methods, it can choose from three options:

1) Inherit the behavior with no change
2) Refine the behavior for itself by changing or replacing the method
3) Disallow the behavior

In addition each new type can add properties and define new methods that are unknown to its supertypes.

An object in ATIS is represented by its element-id or handle for the object uniquely identifying it to the system. Each relationship is a initial class object in ATIS. The base relationship type represents the properties that are common to all relationships and that can be inherited form this type. The type relationship defines the owner and the member to allow the relationships to be directed. Relationships may be defined as one-to-one, one-to-many, or many-to-many.

In ATIS, VERSION is an abstract type that is used to represent any physical object for which history is to be maintained. VERSION keeps track of specific instances of objects and the relationships between these various instances.

The configuration management layer introduces the concept of composite elements, which are sets of elements with some constraints on their composition. The object type COLLECTION describes a composite object and can contain only versionable objects. Collections are associated with a directory in the native file system where the contents of object of the collections can accesses through standard tools.

The ATIS work flow model revolves around pieces of a software systems migrating from the lower level of approval to higher levels through a promotion process. This process requires approval of one or more members of the organization that is responsible for a particular level of approval. As a version is promoted higher and higher through the levels of approval, it is assumed to be more stable, (i.e., the version has passed more acceptance testing or reviews and is therefore more bug-free. Changes at higher levels are made only through rigid approval and tracking mechanisms and there are fewer versions seen at higher levels.

The ATIS reference model, even with its shortcoming and lack of granularity, is used a a framework to measure the features provided by other proposed standards for information resource dictionaries.

4.3.2.2 ANSI X3.138

ANSI X3.138-88 is the American National Standard for Information Resource Dictionary systems, approved in 1988. The ANSI X3.138 uses the entity-relationship approach for data modeling to represent the Information Resource Dictionary (IRD) data. Named entities represent the objects in the system. Entities are types; the attribute and relationship types an entity may be associated with are defined by the entity's type. Entity types are represented as instances of the meta-entity ENTITY-TYPE [BEY90].

Relationships between objects are supported directly, by allowing the client to manipulate relationships as a distinct concept. Relationships are typed; the attribute and entity types a relationship may be associated with are defined by the relationship's type. Relationship types are represented as instances of the meta-entity RELATIONSHIP-TYPE [BEY90].

Attributes are represented as named values associated with either entity or relationships. Attributes may be singular (they have one value) or plural (they may have multiple values. Attribute types are represented as instances of the meta-entity ATTRIBUTE-TYPE. Attribute types are independent of any entity or relationship type; they may be defined before they are used, and an attribute type may be associated with many entity or relationship types [BEY90].

Entity and relationship types may be grouped into structures called IRD schema structures that contains the meat-entities and meta-relationships they are associated with. New types are added to the system by defining new instances of ENTITY-TYPE, RELATIONSHIP-TYPE, and ATTRIBUTE-TYPE, associating them with the schema [BEY90].

The entities in ANSI X3.138 are identified by their access names, the full entity name including revision number and variation name; there is no other realization of object identity. Relationships are represented explicitly. Relationships may have attributes associated with them. Exactly two entities must be associated with each relationship instance; relationships are binary. Relationships with more than two participants must be represented using a junction entity [BEY90].

The behavior of an object is specified by a set of standard operations: read, write, modify erase, rename, and change partition. The concept of the object is not addresses by ANSI X3.138 [BEY90].

The versions of objects in ANSI X3.138 are identified by version numbers and variation names. Version numbers are assigned consecutively from 1, and may not be changed once an entity version is created. An implementation of ANSI X3.138 is assumed to provide low-level, transparent locking mechanisms sufficient to prevent corruption of the information resource dictionary when multiple users are performing updates at one time. Configurations are addressed in ANSI X3.138 in the most general entity type called SYSTEM - an entity which represents a version of a software system [BEY90].

ANSI X3.138 represents the work flow through the concepts of partition, life cycle phase, and life cycle phase class. Life cycle phases are related to one another in a tree structure. The work flow model restricts the representation of configurations by introducing the concept of phase-related relationship types [BEY90].

4.3.2.3 IRDS

Since 1981, the National Bureau of Standards (NBS) of the United States Department of Commerce, through its Institute for Computer Sciences and Technology (ICST), has been regularly conducting a committee whose objective has been to develop a Federal Information Processing Standard (FIPS) for data dictionary systems. The membership of this committee has consisted of representatives from NBS, many federal government agencies, a number of data dictionary vendors, users without vested interests, and in-between groups such as consultants.

In 1980, the American National Standards Institute (ANSI) also started a committee (X3H4) whose objective was to develop a standard for the Information Resource Dictionary System (IRDS). Because the objectives of the two committees were similar, they merged their efforts in September 1983.

The Information Resource Dictionary Systems (IRDS) being developed by the National Institute of Science and Technology supports the design of efficient programs and databases that share data. Currently, the ANSI/IRDS X3H4 was approved as an ANSI standard in October 19, 1988, and is now in maintenance. Plans for future include the following versions [IWC90]:

- X3H4.1 - The objective is to develop a reference model for IRDS. The draft was reviewed in March, 1990, with plans to be completed by June 1990.
- X3H4.2 - The objective is to develop an IRDS services interface to external software and extensions for object-oriented tools. The initial public review was completed in January 1990 for the IRDS Services Interface, a base document for extensions that were reviewed in March, 1990. Plans are the IRDS Services Interface to be completed my May 1990, and extensions in June 1991.
- X3H4.3 - The objective is to develop a functional data interchange file format for import/export. The public review is currently in progress at the end of 1990.
- X3H4.4 - The objective is to develop local naming conventions and name administration for entities described in IRDS. The technical report was reviewed in May 1990, and planned for completion in November 1990.
- X3H4.5 - The objective is to develop an understanding of the issues involved in integrating an IRD schema from an external source into a local IRD schema. The problems, issues, and alternatives have been review, and a technical report is to be completed by May 1991.
- X3H4.6 - The objective is to clarify the role of an IRDS in support of managing the meta data requirements of a distributed, heterogeneous information environment. A preliminary model and annotated outline for the technical report has been develop, and the technical report is planned to be completed by December 1990.

A data dictionary is an information system describing an application's significant data. It does not store the organization's data like a database management system (DBMS). Instead, it contains information about the data used by an organization. The IRDS defines the functionality of a standard data dictionary system plus a system-standard schema. The system-standard schema identifies a set of entities, relationships, and attributes which must be included in all conforming IRDSs. The system-standard schema includes three categories of entities: data, process, and external. Data entities include document, file, record, element, bit-string, character string, fixed-point, and float entities. Process entities include system, program, and module entities.

The only external entity is the user. In addition, standard relationships and attributes are defined for these entities. The system-standard schema can be extended to particular applications and installations through the addition of other entities and relationships. The significance of the IRDS on frameworks is that it represents a data model and DBMS independent vehicle for describing data, sharing data, and supporting tool interoperability [RUD89].

The Information Resource Dictionary Systems (IRDS) is a proposed ANSI/ISO standard for a data dictionary, and is the Federal Information Processing Standard (FIPS) for data dictionary systems. Used to capture metadata during the system life cycle, the IRDS is being developed by the National Institute of Science and Technology and supports the design of efficient programs and databases that share data. The significance of the IRDS for frameworks is that it represents a data model and DBMS independent vehicle for describing data, sharing data, and supporting tool interoperability.

An IRDS implementation which conforms to the standard must provide either a command line or panel interface. An implementation may also provide a service (callable) interface, however, there is no standard yet for a services interface. IRDS supports an extensible entity relationship model for storing metadata. IRDS supports a four level view of data in a system:

- Schema Description Layer
- Schema Layer
- Metadata Layer
- Data Layer

The lowest level is the physical database data; each higher level describes and controls the format of the layer below it.

The IRDS **Schema Description Layer** describes and controls the basic elements which make up the schema language of the IRDS. The IRDS Schema Description Layer defines the behavior and meaning of entity, relationship, attribute, and attribute group types. This layer is supported by the IRDS implementation.

The IRDS **Schema Layer** defines and controls the schema of the metadata stored by an IRDS. In an application, this layer is used to define the entity and relationship types which are required to store the information of interest (i.e., user account data, the local schemas, the network schema, the network schema views, the custom schema view, and the correlations). Entity and relationship types are defined which specify the attributes and connections needed to represent a local database entity type definition and to link it to the

local database schema to which it belongs. This layer is defined by the application and loaded into the application IRDS when an IRDS is created.

The IRDS **Metadata Layer** describes the actual data stored in the physical database. In the application, this layer contains the various schemas and views which will be used to access the physical databases. Correlations, the user account, and the user privilege are stored in this layer.

The IRDS **Data Layer** represents data in existing databases which is to be accessed by the application. The database may be created specifically for the application or may be created for other purposes.

The proposed standard for the IRDS provides both a core model and rules for extension to represent the schemas, dictionaries, and encyclopedias of instances of models in the Entity-Attribute/Relationship-Attribute form. Heterogeneous computing sites that support the standard can interact remotely and exchange, and using such information as is authorized. At a minimum, the standard can scale down easily to represent the dictionary and schema requirements normally associated with relational data base management systems. At best, the core model can be legally extended to support a common approach to object-based modeling and management of systems, software, hardware, and human factors concerns (both technical and management).

The user payback for the IRDS is through the storage and management of components of application development in one logical dictionary system which can be a cornerstone for automation and reusability in the future. Global dictionary systems will serve the organization as a whole, which will feed the run-time data dictionary systems which are coming into widespread use under fourth-generation technology. Many productivity tools have had dictionary capabilities imbedded within them in the past and will continue to do so in the future. However, it is essential that this capability is moved from strictly a proprietary status to a shared status so that integrated environments can be satisfactorily built in the future. Likewise, in order to provide stability for future user requirements, it will be essential that metadata can be easily migrated from one dictionary system to another by use of an interface within the IRDS standard. The IRDS provides a mechanism to allow vendors' products to be engineered together into total solutions resulting in a payback for the vendors. The IRDS standard as well as other industry standards can provide a very realistic way for vendors to segment the market for application development and maintenance products [ACL87].

The ISO IRDS is the committee draft, as of September 1990, of the proposed IRDS Services Interface being developed by the IRDS Rapporteur Group of

OSI/IEC JTC1/SC21 WG3 (International Standards Organization/International Electrotechnical Commission Joint Technical Committee 1/Subcommittee 3 21 Working Group 3) under project JTC 1.21.6.2., as defined by WG3 document number IOS/IEC JTC1/SC21 N4895 [BEY90].

The ISO IRDS uses the relational approach to represent IRD data, as defined by the SQL standard. Objects, their types and attribute values are represented as rows and columns in a table. New types are added to the system by adding rows to the tables. the ISO IRDS organizes types into schema structures as in the ANSI X3.138.

Operations that can be performed on objects ("read", "create", "modify") are more limited than ANSI X3.138. The content of the object is not addressed by ISO IRDS. ISO IRDS handles versioning by introducing working sets and working set versions. A working set version is a single unit for the purpose of change management. It is possible to specify that a working set version is linked to a previous working set version. This inheritance can also be disabled. The ISO IRDS does not define any facilities for controlling simultaneous update to a tree of working sets by multiple users. However, the working set version concept does assist in supporting multiple users by allowing a project to be broken up among multiple working sets which may be updated independently. A work set is controlled by exactly one user, who may grant various levels of access to other users, permitting some degree of control over updates to the IRD.

Configurations are defined in ISO IRDS by allowing one working set version to cross-reference the definitions in one or more other working set versions in addition to the one on which it is based. So while an object version is allowed to exist in only one working set version , it may be referenced from any number of other working set versions, not necessarily belonging to the same working set.

ISO IRDS adopts a similar work flow model to X3.138, adapting it to the needs of its version and configuration models. More than one version of the same object may exist within a single life cycle phase.

4.3.2.4 PCTE and PCTE+

The Portable Common Tool Environment (PCTE) is a software engineering framework that provides a standard set of tool support layer interfaces. PCTE+ is the militarized version of PCTE. PCTE has been developed as a result of an initiative sponsored by the Commission of the European Communities (CEC) under the auspices of its ESPRIT research and

development program. The PCTE strategy is to provide an efficient and powerful platform to support CASE development and integration. The PCTE interface specification defines a public tool interface for promoting tool portability, and this specification is in the process of being standardized. The PCTE provides the capabilities of a virtual operating system, an object management system, and a standard set of high level user interface services. The PCTE is designed to operate in a distributed computer network providing transparent data distribution and process management [RUD89]. PCTE distribution services provide access to information items over personal workstations and servers in a LAN (local area network) through ISO/OSI standard communications services.

The PCTE interfaces support entity-relationship project databases, user interface services, and process and transaction management [RUD89]. PCTE processes are modeled as entity-relationships, not objects. Consequently, the PCTE provides a separate set of interfaces to manage processes. This approach is difficult when writing a generic tool to uniformly access a wide variety of objects managed within the framework. Another problem occurs when a process is terminated. When this happens, all information about the process is lost. PCTE+ has proposed a change to the PCTE model to make processes first-class objects [RUD89].

PCTE processes are initiated in the context of an activity (i.e., transaction). Acquired resources and locks held on these resources are associated with an activity. An activity can be initiated internally to a single process or extended to include several generations of processes. Consequently, the PCTE model binds processes to activities. Ada tools that use tasking to perform separate functions as transactions would function improperly if the Ada run-time manages tasks within a PCTE process [RUD89].

Experience with the PCTE has precipitated a growing body of experienced developers who have tried to integrate tools into advanced frameworks. For example, the ESPRIT PACT project found the need to develop a layer on top of the PCTE interfaces and use these new services for developing tools. One of the problems the PACT developers found when they started to develop tools using the PCTE was that the PCTE interfaces were at too low a level. Consequently, the PACT project introduced another layer called the PACT Common Services layer.

The Common Services layer implemented some of the fundamental models that the PACT project wanted to reflect through all of the tools used within the environment. By factoring out this functionality and embedding it into a Common Services layer, the tools became simpler and it became easier to provide a single, consistent model of user interaction [RUD89]. The Common

Services layer eases the tool writer's task by reducing the amount of code to be written. It also provides a higher degree of consistency amongst tools. The PACT project found the Common Services layer to be an important mechanism for tool integration support and recommended that these services be used by all tools.

PACT's experience with PCTE introduces an interesting dilemma in that the level of tool portability is now the Common Services layer and not the PCTE layer. In order to transport tools, it is also necessary to transport the common services layer. Furthermore, it is unclear if tools built to the PCTE interfaces could be integrated with tools developed on top of the Common Services layer.

Nonetheless, PCTE implements a well suited strategy to support easy migration from UNIX by providing a compatible emulation of the "system calls" functional level of UNIX System V [NAS85]. For its wide availability, UNIX is expected to play the role of the "initial common environment" for the present time.

In PCTE+'s Functional Specification Issue 3, October, 1988, the data modeling approach is type driven. It is described in terms of an entity-relationship-attribute model, and does not use the object-oriented paradigm. Types in PCTE+ are defined in a multiple inheritance hierarchy, but these types have no associated semantic behavior. PCTE+'s object layer contains a schema management system that defines the types that are available in the system, and an object management system that describes the objects that are in the system.

Objects are defined in the traditional Entity-Relation sense. All objects in the system have a system identifier as well as a type that uniquely defines the object. Relationships between objects are assumed to be bi-directional. Objects can be designated by pathnames and references. The behavior of an object in PCTE+ is fixed and defined by the standard and include those to build a schema, manipulate, link, and relate objects and their attributes directly or by reference, and manipulate the contents of objects.

PCTE+ does not have a version management layer as such, but does provide mechanisms on which a version model can be built. PCTE+ defines a relationship to link objects together and then overloads that relationship with versioning semantics. Relationships may have two links: a predecessor and a successor. Two components that were previously separate can be merged into a single component. Also, a single component may be split into two components, such as the separation of the specification and body of an Ada package.

The PCTE+ configuration model relies on composite objects that are formed by connecting two or more objects together by links. Objects can be components of multiple composite objects. The PCTE+ does not address work flow, and can be seen as lacking when compared to other existing standards and standards work in progress.

ECMA TC33-PCTE was formed to develop common tool interfaces for a wide range of operating environments to ensure a foundation for portable interoperability of tools. Focus is on the mechanisms to provide common functions between tools using PCTE+ issue 3 as a base start.

4.3.2.5 CIS

Begun in 1987, the effort was originally called ATIS (Atherton Tool Integration Services), a joint initiative by Atherton and Digital Equipment Corporation to develop CASE interface standards through industry participation. Initial members included platform vendors such as IBM, DEC, HP, Sun, and Apollo, and CASE vendors such as Atherton, Cadre, IDE, and Softool. Three working groups formed, one to focus on requirements for integration services, a second to look at standards for tool portability, a third to examine control issues such as tool invocation [ST89b]. The CASE Integration Services (CIS) grew out of ATIS. ATIS is the current draft of the ANSI work towards the next version of the IRDS standard, as defined by WG3 document number ISO/OEC JTC1/SC21 WG3 N1020.

CIS was an industry effort to develop and promote CASE interface standards for tools with emphasis on tool to repository and tool to tool interfaces dealing with computer services, not semantic models. This effort worked in conjunction with the Software Productivity Consortium and was an ad hoc group of system vendors, independent software vendors, CASE tool users and research consortia.

The motivation for the effort was Digital's perception that large aerospace corporations were looking for fully populated IPSEs (Integrated Project Support Environments). Using the Atherton Software Backplane as a starting point, Digital and Atherton drafted an initial ATIS proposal which was presented at the PCTE workshop in June 1988. The ATIS proposal was again presented at the U.S. CASE Expo in Dallas, May 1988, at the CASExpo in Cambridge, 1988, and in San Diego, September 1988. The primary issue at this point is the relationship ATIS has with PCTE and CAIS-A.

The principal area of focus of the CIS effort was in the area of control integration, even though some participants advocate that they do not

overlook data integration. The CIS effort focused on object-oriented link services and message switching, and looked beyond both the CAIS-A and PCTE work towards the use of object-oriented CASE databases and control models. The CIS participants included Apollo, Aerojet, Atherton, Cadre, Contel, Digital, Ford Aerospace, Hewlett-Packard, IDE, IBM, McDonnell Douglas, Olivetti, SEI, SPC, and SUN Microsystems. Apollo lead the participants in this effort. Most of the industry participants took the position that object-oriented frameworks will be the driving technology of the 90's [RUD89].

CIS planned to release an explicit model for comment in late 1990 with prototype services available by June 1991. Instead, the group reorganized as an ANSI standards group called X3H6. As of July 1992, X3H6 is focusing on object-oriented environment models (i.e., classes and methods) and standards for open systems.

4.3.2.6 PCIS

Begun in 1989, efforts continue to bring about a common Europe/United States Standard interface for the support of Ada programming tools. The Portable Common Interface Set (PCIS) will be a single interface standard based on the convergence of the Europe's Portable Common Tool Environment (PCTE+) and the U.S.'s Common Ada Programming Support Environment (APSE) Interface Set (CAIS-A). PCIS has been described as a progressive evolution from CAIS and PCTE that incorporates soundly based emerging technologies while providing a low cost migration path to support integration of programming tools.

The PCIS Programme is designed to define the PCIS, assess it, and introduce it into use. The essential core of the PCIS Programme's work will be in identifying the features required for a PCIS based upon CAIS-A and PCTE+. The PCIS Definition Phase will produce the PCIS Abstract Specification, an Ada Binding, a C Binding, a PCIS Reader's Guide, and migration guides. This phase will be managed under the auspices of the Special Working Group (SWG) on APSE. SWG on APSE participants include Canada, Denmark, France, West Germany, Italy, the Netherlands, Norway, Spain, the United Kingdom, the United States, and the NATO Communications and Information Systems Agency (NACISA).

The most recent development was in May, 1990, when the Ada Joint Program Office (AJPO) announced that it was seeking experts on the Integrated Project Support Environments (IPSEs) to represent the U.S. on an international team that will support the Definition Phase of the PCIS Programme [ST90b]. The IPSE has evolved to meet the needs of both DoD and the commercial sector.

Public reviews are planned in 1992 and another in 1993. It is anticipated that a final PCIS specification will be available by mid-1994, and a PCIS-based environment will be available by the late 1990s. The PCIS specification will ultimately be submitted for acceptance by the International Organization of Standardization (ISO).

4.3.2.7 CAIS-A

The Common APSE Interface Set Version A (CAIS - A) is a completed DoD standard for the interfaces of the tool support and data management layers of a framework. The mission of CAIS-A was to develop a common Ada project support environment (operating system) interface standard for Ada applications development. The CAIS/CAIS-A provides an integrated view of the user's access to the tools and other services, and resources of the environment via their terminal or workstations. The CAIS-A specifications were produced under contract to NOSC and were developed and reviewed by the KIT/KITIA, a combined effort among Government, industry and academic representatives. KIT/KITIA completed its mission in 1987, but ramifications of its work still reverberate at AJPO. The group's goal was to define a set of framework interfaces to promote tool transportability.

CAIS - A, the proposed set of Ada interfaces and interface semantics for the tool support layer, provides a modified entity-relationship project database with multiple inheritance, hierarchical transactions, mandatory security, and naming flexibility. The CAIS-A interfaces support a variant of an Entity-Relationship data model, a process model, interprocess communications, and input/output. CAIS processes are objects whose attributes contain process status information and whose relationships establish the context of the process relative to other processes and objects within the environment. The existence of process nodes is independent of whether they are running, suspended, or aborted [RUD89].

The CAIS-A design addresses several problems that have not been addressed to the same degree in other frameworks. First, CAIS-A provides a uniform object model for data, processes, interprocess communications, and security. Second, CAIS-A addresses name conflict resolution through a flexible view mechanism. The PCTE has subschemas that provide similar capabilities. The CAIS-A approach is more robust and type-safe. CAIS-A addresses both discretionary and mandatory security and provides a more flexible, discretionary security model. This approach is different from the PCTE model which is more fragmented. PCTE processes are not viewed as objects, although this deficiency is being addressed by the PCTE$^+$, the militarized version of PCTE.

CAIS-A is a very sophisticated interface set that would require a substantial investment to implement as a production quality framework. At this time, CAIS-A does not provide interfaces to support user interface development. Prototype implementations of CAIS - A are under development, and the CAIS-A prototype developed by SofTech is substantially complete. The effort to implement CAIS-A has been estimated at 35 to 65 man years.

CAIS-A is far more powerful than the original CAIS. The original CAIS, used with IRDS, can be used to create the lowest layer of the persistent object base to support the phases and activities with stable interface sets. Since entities can be mapped to passive objects, neutral objects, or active objects, two immediate benefits are now available. First, all tools and devices under baseline control in the environment can be represented, along with their legal relationships and properties, in the same dictionary and encyclopedia format used for the models of the various projects to be supported by this environment. The fine granularity permitted by the common dictionary and encyclopedia representation permits knowledge-based technology to be applied to the models of the application projects.

4.3.2.8 SQL

The Structured Query Language (SQL), designed for relational databases, is the standard block-structured language for defining and manipulating instances in a relational database. The SQL standard defines the logical data structures and basic operations for an SQL database. It provides functional capabilities for designing, accessing, maintaining, controlling, and protecting the database.

SQL was originally developed at IBM in System R, the relational database management system. The American National Standard Database Language SQL specifies the syntax and semantics of interfaces to a database management system for defining and accessing SQL databases. Together, these interfaces are called Database Language SQL.

The SQL standard was developed by the Technical Committee on Database, X3H2, under project 363D authorized by the Accredited National Standards Committee on Information Processing Systems, X3. The purpose of this project was to develop a standard for the functionality of the interfaces (i.e., the set of functions and the semantics of individual functions) to a relational database management system. These functions will be used in defining, querying, and altering relational databases. The SQL standard was approved as an American National Standard by the American National Standards Institute on October 16, 1986 [AME86].

The SQL standard specifies the syntax and semantics of two database languages:

- A schema definition language (SQL-DDL) for declaring the structures and integrity constraints of an SQL database
- A module language and a data manipulation language (SQL-DML), for declaring the database procedures and executable statements of a specific database application program

The primary reason for adopting SQL is to promote transportability and interoperability of database applications. This SQL standard provides a vehicle for portability of database definitions and application programs between conforming implementations. In the software engineering environment domain, there are many tools that employ relational databases. Project, quality, and configuration management tools are of particular interest. Tools that adhere to the SQL standard should be able to interface to different SQL compliant databases.

By promoting transportability and interoperability of database applications, SQL circumvents the need for every PC database to be file-compatible with every mainframe database. This eliminates the conversion task which would cost more than the mainframes themselves. SQL is a simple programming language which produces programs to query minicomputer and mainframe databases. Modern databases rigorously separate the data which they store from the programs which are used to retrieve it. This means that a variety of data extraction programs can be used on the same data files, and SQL is one of these extraction programs. If a PC database includes SQL, its queries will be treated in the same way by any SQL-compatible database on any machine which receives them. With these advantages, various mainframe and microcomputer software companies have either launched products supporting SQL or announced support for it in future products [JAC87].

SQL, based on a relational model, is easy for both users and developers to learn and understand. Developers can communicate and implement user needs concisely, and users can create, test, and perfect queries as a team. The SQL is appealing to developers because prototypes are built quickly and can be designed to fit user requirements more precisely. In addition, SQL provides the capability to change database products without an application impact [CAA89].

The SQL standard applies to implementations that exist in an environment that may include application programming languages, end-user query languages, report generator, systems, data dictionary systems, program library

systems, and distributed communication systems, as well as various tools for database design, data administration, and performance optimization.

Additions to the SQL are planned to include referential integrity, enhanced transaction management, specification of certain implementor-defined rules, enhanced character handling facilities, and support for national character sets [AME86].

Of the relational database systems available today, there have been significant extensions made to the SQL standard. Independent of these problems, the Ada community has been working for several years to define Ada bindings to SQL. SQL is one way to standardizing an Ada binding to a relational database subsystem interface. However, this effort has difficulties in overcoming the scaling direction problem (i.e., either up or down). Announcements have been made in 1989 concerning the availability of commercial SQL products with Ada bindings for use from within Ada programs [RUD89].

4.3.3 Products

4.3.3.1 AD/Cycle

The Application Development/Cycle (AD/Cycle) is an expansion of IBM's Application Development/System Application Architecture (AD/SAA) to encompass the entire development life cycle, including maintenance. On September 19, 1989, IBM announced the AD/Cycle and shook an industry that had been struggling to establish its own legitimacy. IBM waited one year before announcing their repository to the industry. Most CASE vendors and many prospective users want to wait before making any strategic decisions [ST90a].

AD/Cycle is completely compatible with SAA, and is positioned as the second major SAA application (after Office Vision). For consistency, all tools (IBM or vendor) will follow IBM's Systems Application Architecture (SAA) Common User Access (CUA) guidelines. CUA is available primarily through PS/2, OS/2, and the Presentation Manager. In order to maintain their profits, IBM will declare certain components of the AD/Cycle exclusive domain, (i.e., the repository and its underlying database (DB2), and the code generator technology (CSP) as proprietary property. So while AD/Cycle is open to other tool contributors, IBM will hold the reins firmly where the evolution of the architecture and interface definitions are concerned.

Not all vendors are created equal in the open AD/Cycle scenario. Selected business partners will have definite marketing priorities and early access to information, including the evolving meta model and tool interfaces. Even

though some vendors are favored by IBM, other CASE vendors do not plan to dump their own dictionaries and encyclopedias and work only through IBM's Repository Manager. A likely scenario would be that most CASE vendors will have their own tool-specific repository as proprietary with an export facility compatible with the IBM Repository Manager. This will be particularly true for vendors offering integrated workbenches. Others will offer stand-alone repositories as separate product offerings. Their competitive advantage will be based on earlier availability, added functionality, support for official industry standards, and the ability to operate in a multi-vendor hardware environment.

IBM's AD/Cycle vision will affect most, if not all, of the information systems community. It signifies IBM's support of the CASE industry, and signals the beginning of widespread CASE tool integration. By making this announcement, IBM has acknowledged the need for a more disciplined approach to software development and a much higher level of tool technology to support new software engineering practices. In addition IBM has taken an open systems approach to the CASE environment by promising to make public the tool interfaces and the repository meta-model. Going one step further, IBM will provide training for CASE vendors waiting to integrate their tools into the AD/Cycle framework.

IBM envisions AD/Cycle to be an architecture, a philosophy of open framework composed of two distinct parts: life cycle tools, which will share application development information through the life cycle, and an application development platform, which will provide services for the integration of the tools. The planned AD/Cycle incorporates an integration platform based on a cooperating enterprise processing model, including analysis and design, application production composed of languages, generators and knowledge-based system, testing and maintenance, a common user interface for all tools, and a set of technical and cross life cycle tools from IBM and selected CASE business partners. Tool services will include library functions, administrative functions, standard operations, such as copy, delete, move, and store, and policies of the Common Programming Interface (CPI).

As a central focus of the AD/Cycle framework, the repository acts as the information base, controlling administration, definition, storage and retrieval of all application development information. As a part of the AD/Cycle, the Repository Manager provides a CASE enterprise database, a meta model, and tool integration services. When it is available, it will be a host-based DB2 implementation and the primary vehicle for sharing application development information among the tools.

AD/Cycle is a repository designed to assist information systems professionals by improving the productivity, quality, and manageability of the systems creation and maintenance process. AD/Cycle supports process improvement, and indirectly assists customers in reducing risk, decreasing the cost, improving the first-time quality, and decreasing the time to completion in their systems efforts.

The PS/2 workstation is the primary window into the AD/Cycle environment. It will operate from an IBM-recommended minimum workstation Model 70, or above, with INTEL 80386 or 80486 processors and 12 megabytes of memory, and 115 megabytes of fixed disk with OS/2 Extended Edition connected to a host. This is viewed as a cooperative approach in application development.

The information model, with its three tiers, enterprise, design and technology, provides structure and format definitions for information in the repository. The complete implementation will be phased and evolutionary, and should be extended to allow integration of future tools into AD/Cycle.

For the future, when the Repository Manager is fully integrated, individual tools will not only put design information into the repository for code generation and archiving, but these tools could share and cross check their respective design representation to improve the level of automation and ensure higher quality.

Over the long run, IBM is expected to provide more and more of the components of the total CASE environment. However, until the technology matures, something that is not foreseen in the next decade, many CASE vendors will thrive in coexistence with the AD/Cycle by providing leading edge technology and by addressing the special needs of targeted user segments.

Some components of the AD/Cycle are available today, others will be delivered in a staged process in the next 1-2 years. Until the components arrive, the ramifications of the AD/Cycle announcement remain unknown. Only when AD/Cycle is in place and in use can we truly determine whether it is the CASE 'solution' for our industry today [ST90a].

Forte sees that the dynamics of the technology and the industry will initially result in a two-tiered repository structure [FOR89b]. Groups of integrated tools from one or more vendors will use an internal repository that provides the performance and unique facilities needed to support the toolset's competitive edge. These local repositories will interface to AD/Cycle's

Repository Manager or another repository product to provide coordination across projects and throughout the enterprise.

4.3.3.2 Network Database Language

In March 1987, the American National Standard Database Language, NDL Network Database Language, ANSI X3.133-1986, was adopted as a Federal Information Processing Standards (FIPS). The Network Database Language (NDL X3.133-1986) is available from ANSI and is related to the POSIX 1003.1 standard [IEEE88]. The ANSI X3.133-1986 specifies three languages that make up a network model database management system [USA87]:

- A schema definition language for declaring the structures and integrity constraints of a network structured database
- A subschema definition language for declaring a user view of that database
- A module language including NDL statements, for declaring the database procedures and executable statements of a specific database application

The purpose of this NDL standard is to promote portability of database definitions and database application programs between different installations. The standard is used by implementors as the reference authority in developing a network model database management system and standard language interfaces to the database management system; and by other computer professionals who need to know the precise syntactic and semantic rules of the standard [USA87].

Federal standards for database management systems should be used for computer database applications and programs that are either developed or acquired for government use. The FIPS Database Language NDL (FIPS NDL) is one of the database management system standards provided for use by all Federal departments and agencies. FIPS NDL is suited for use by applications written in one of the FIPS programming languages for which NDL module language is specified in ANSI X3.133-1986; e.g., COBOL, FORTRAN, or Pascal [USA87].

The FIPS NDL is suited for use in database applications that employ the network data model. The network data model is appropriate for highly structured applications requiring rapid access along predefined paths. Although this standard does not specifically address distributed database applications, it may be used, along with facilities for distributed transaction processing, to access network structured data at remote nodes in a distributed system [USA87].

The use of FIPS database languages is strongly recommended for database applications when one or more of the following situations exist [USA87]:

- It is anticipated that the life of the database application will be longer than the life of the presently utilized equipment or database management system, if any.
- The database application is under constant review for updating of the specifications, and changes may result frequently.
- The database application is being designed and developed centrally for a decentralized system that employs computers of different makes and models or database software acquired from a different vendor.
- The database application will or might be run on equipment other than that for which the database application is initially written.
- The database application is to be understood and maintained by programmers other than the original ones.
- The database application is or is likely to be used by organizations outside the Federal Government (i.e., State and local governments, and others).

The NDL standard does not specify the following [USA87]:

- Concurrent processes accessing the database by multiple users or processes
- The limits on the number or sizes of database constructs; e.g., subschemas, records, sets, components, boolean expressions, etc.
- Application pre-processing facilities for producing separate standard database modules and standard language programs
- A distributed database facility

NDL is one of the components of the Computer-Aided Acquisition and Logistics Support (CALS) Standard Framework for the Integrated Information Support System.

4.3.3.3 CODASYL

The Conference on Data Systems Languages (CODASYL) is the group that developed the COBOL language and also established a Data Base Task Group (DBTG) to develop a database model for processing using the COBOL language. Such a model was developed and submitted to the American National Standards Institute (ANSI) for acceptance as a national standard. The DBTG database model is important not only as a national standard, but

also because there are several commercial database systems based on it. The Integrated Information Support System (Air Force Systems Command) includes CODASYL as one of the components of the CALS standard framework.

Like SQL, CODASYL's DBTG model has a DDL, a data definition language, and a DML, data module and manipulation language. The DDL is used to describe the database schema and subschema. As a description of the schema, the DDL is intended to stand on its own and be independent of any programming language. As a description of a subschema, however, the DDL is intended to be an extension to COBOL. The DBTG left the responsibility of other groups to develop a subschema DDL for other languages. The DML of the DBTG model, also an extension of COBOL, is a complex array of intricate commands to operate on the database as defined by the schema and subschema. The DDL identifies definitions for specific data structures as summarized in Table 4.3.3.3-1.

Table 4.3.3.3-1. Summary of DBTG Data Constructs [KRO77]

Data Structure	Description
Data-item	Unit of homogeneous data; corresponds to field
Vector data-aggregate	Array of homogeneous data-items
Repeating group	Collection of data-items or aggregates occurring multiple times
Record	Collection of data-items or aggregates; logical concept, not a physical one
Set	Collection of records
Set member	Record that belongs to a set
Set owner	Record that identifies a particular group of set members (set occurrence)
Realm	Subset of database records

The DBTG model can represent record relationships with tree and simple networks. Complex networks cannot be directly represented, but they can be transformed into simple networks for representation. The set is the key concept for modeling relationships in the DBTG. Important characteristics are summarized [KRO77]:

1. A set is a collection of records.
2. There are an arbitrary number of sets in the database.
3. Each set has one owner record type and one or more member record types.
4. Each owner record occurrence defines a set occurrence.
5. There are an arbitrary number of member record occurrences in one set occurrence.
6. Set records can be ordered.
7. Set records can be accessed directly by specifying the values of record data-items.
8. A record may be a member of more than one set type.
9. A record may not be a member of two occurrences of the same set type.
10. A record may not be a member and an owner of the same set type.

The DBTG model does not prohibit cycles. They are easily represented with the set construct and do not violate any of the DBTG set concepts. The DBTG left it to implementors of the model to decide whether or not cycles were permitted in their implementations.

Critics of the CODASYL database say that the system does not satisfy some of the typical rules for describing an object-oriented database. CODASYL supports complex objects and object identity to a partial degree; however, CODASYL does not support encapsulation, types or classes, inheritance, binding, extensibility, persistence, concurrent users, recovery protection, and a simple way of querying data [ATK90].

On the other hand, the DBTG model in a CODASYL environment helps to cope with the rapid growth of size and complexity of databases in the recent years. Several methodologies for database design have been proposed, but manual design methodologies are not powerful enough to cope with present design requirements, characterized by semantic complexity and stringent user requirements with respect to data availability and application performance. Many research efforts have concentrated on automated database design tools based on the CODASYL environment [ORL85]. In addition, research continues in providing a transition path by a standard interface for large CODASYL database systems away from COBOL and into Ada to support the Ada mandate for use in U.S. Department of Defense mission critical computer systems [MCN86].

4.3.3.4 Cohesion

Cohesion, Digital Equipment Corporation's new integrated CASE environment, is based on DEC's repository product, CDD/Plus and its associated tools services strategy, Application Tool Integration Services (ATIS). All are currently available as products.

The Cohesion name was chosen to emphasize the characteristics of DEC's software development environment, which supports software deployment on heterogeneous systems from a unified environment; a framework that integrates the entire software development life cycle, beginning with enterprise planning through maintenance and on-going management; flexibility of a development environment that works across all industries and businesses; and linkage of business planning with software development, supporting the goals of the entire organization, as well as individual departments. DEC announced several enhanced service and product capabilities of its own and from other software vendors that are all part of the Cohesion environment and support NAS services. DEC suggests that adopting an enterprise-wide computer-aided software engineering environment can result in higher productivity and lower costs if the tools are used and maintained efficiently. DEC promises assistance, via comprehensive training and support, throughout the complete process.

DEC's architecture has four components: services, standards, integration framework and tools. In the standards area, DEC is committed to using standards where they are available and championing efforts to create standards where they are not. The repository is included in the integration framework where DEC provides its own Information Model for data and process. At the tool level, DEC intends to offer products that cover the entire life cycle, such as the methodology tools from Arthur Anderson, Chicago. Cohesion is built on DEC's NAS Network Application Support and is designed to provide the framework for the development of NAS applications as well as software for IBM mainframes, a range of microprocessors and super computers [BER90].

Presently, Cohesion is based on CDD/Plus, DEC's relational database, Rdb, and contains an IRDS-compliant E-R interface. A stated goal for the product is to move to an object orientation which will be available in the next release of their database called CDD/Repository. CDD/Repository is planned for both VMS, UNIX, and Ultrix platforms. It is in test and is planned to be available in the first half of 1991.

DEC's strategy for developing an all-encompassing environment for the production of software is similar to that of IBMs. Opinions differ on the choice of the best implementation of these similar strategies. For example, Tim Tyler, Vice President of software planning for SmartStar Corporation, sees that DEC is superior to IBM by offering more facilities for moving between phases of the life cycle. On the other hand, George Colony of Forrester Research Inc., a Cambridge, Mass. market research and consulting firm, sharply criticized Digital's offering. Colony said Cohesion has so far failed to enlist support of the major CASE vendors, who have strongly backed AD/Cycle [COX90].

4.3.3.5 Rational

Rational®, located in Santa Clara, CA, was founded in 1980 to provide advanced software technologies for multiple-site development of large, complex software systems. Rational's primary product, the R1000 Development System, provides an interactive Ada development environment - the Rational Environment™ - that supports the life cycle activities of a software project using the Ada Programming Language.

The initial release of the Environment was developed between 1980 and 1985, using conventional, batch-oriented development tools. At its first release in 1985, the Environment consisted of 800,000 lines of Ada source code. The direct development effort expended totaled 700 person-months. The team grew from 4 to 20 software engineers during the five-year project.

Newer releases now provide interactive syntactic and semantic analysis of programs or program fragments with knowledge of Ada in the Environment. The Environment also embodies knowledge of the structure of systems developed on it, automating many time-consuming and error-prone tasks in designing, developing and integrating a software system. The Environment performs system builds automatically and uses the minimum recompilation after a change has been made. Incremental compilation reduces the time required to make and test changes. The Environment also provides interactive facilities during integration and maintenance. Rational support facilities allow the building and testing of system configurations without copying or recompiling.

The R1000 Development System is a universal host development system that allows software to be developed on the Rational Environment for ultimate execution on a variety of target computer systems. The R1000 Development System and the Rational Environment are part of Rational's approach to

improving software development productivity. This approach includes training, technical consulting, and product support.

The R1000 Development System is available in two configurations, the Series 300 Coprocessor and the Series 300 System. The Series 300 Coprocessor is a diskless configuration that works in conjunction with file servers from IBM, Sun, and DEC. The Series 300 System incorporates its own, Rational-supplied, disk resources (proprietary hardware.). The R1000 Development System Series 300 has a wide variety of system upgrade options to meet the demand of projects as they grow in size and complexity through the software development lifecycle. Rational's pricing structure allows customers to invest progressively as a project requires additional resources. The price for the Series 300 Coprocessor starts at $99,000, whereas the price for the Series 300 System starts at $199,000.

In addition to the development system and the environment, other components of the Rational product family include the following:

- Rational M68020/UNIX Cross-Development Facility
- Rational Design Facility
- Cadre Teamwork Interface
- Rational X Interface (RXI) to DEC VAXstation and Network Computing Devices' Network Display Station
- Rational Remote Compilation Facility (validated Ada compilation system)
- Rational Performance Analysis Interface
- Rational Network Mail SMTP Gateway
- Rational Publishing Interface
- Rational Configuration and Version Control System (CMVC)
- Rational Subsystems for Project Management

Rational's customers are among the largest defense aerospace, and commercial companies in the United States, Europe, and the Asia-Pacific region. Rational has tracked their customers with measurements of the Environment's productivity impact on the lifecycle of Ada development projects. Rational has statistically shown substantial benefits of their products by increasing the productivity of the development teams, improving of the software quality, reducing project and schedule risk, and improving maintainability and ease of software product evolution.

The down side of Rational's product line is that it is a costly investment with few tools other than for the Ada environment. As such it is ideal for Ada, but not applicable to development in other languages.

4.3.3.6 Atherton Technology Product Line

Atherton Technology, based in Sunnyvale, CA, is addressing the emerging CASE market by offering products that focus on integration and automation. Atherton has developed a framework product that when used with applications also developed by Atherton, allows for the integration of commercial as well as internally developed software tools. This framework provides built-in management, data and work flow controls and runs all the popular hardware platforms (DEC, Sun, IBM), preserving users' investments in technology. Atherton's framework and applications provide mechanisms for automating the entire software development process from beginning to end, including the unique administrative policies and procedure designed by each organization to govern software development.

Founded in January 1986, Atherton set out to develop the platform and the tools that would significantly improve an organization's ability to manage software development projects. The company is addressing the market with a new CASE concept called the Integrated Project Support Environment (IPSE). Two additional products work with the IPSE, the Software BackPlane and the SoftBoard Series.

The Software BackPlane is a framework used for building and managing an IPSE for software development. Software BackPlane can be installed on a variety of hardware platforms and then integrated with independent software tools into a consistent user interface. The Software BackPlane enhances the native environment for host computer platforms by providing a data repository, repository services, and a comprehensive user environment. Capabilities include version control, access control, audit trails, and a programmatic interface that includes work flow control, tool integration, and metric gathering and reporting. Software BackPlane provides services to link together and coordinate the wide range of the users' favorite in-house and third-party tools used on software development projects.

Software BackPlane's object-oriented data management facilities provide a consistent, logical view of all project data. The repository controls and coordinates large sets of objects developed throughout the software development life cycle by project tools or by the Software BackPlane. Users can interactively query a selected database and its objects. Available queries include the object's status, history of its evolution, and its relationship to

other objects. Software BackPlane's user environment provides access to all integrated tools, the native file system, on-line help, and printing support.

The SoftBoard Series consists of the Integration SoftBoard™ and the Project SoftBoard™. Integration SoftBoard helps customers create a customized software development environment with the CASE tools of their choice within the IPSE. The integrator's job is easier and faster by providing an object-oriented approach to tool integration and reuse, C-based template for integrating different types of tools, edit, compile, and test loops, and a toolbox of utilities.

The Project Softboard is a project management application for automating structure communication among project team members. Facilities that are provided allow users to customize action request forms, track the flow of action requests between project members, associate action requests to objects in the repository, and query the status of action requests and related objects. Project SoftBoard provides automation and control for formal project communication, useful for change authorizations, bug tracking, and general communication between team members to improve the team's efficiency and quality of their work.

4.3.3.7 SLCSE Database

The Software Life Cycle Support Environment (SLCSE) is a software engineering environment that supports data and process integration. SLCSE, a second generation environment, supports the description of the structure of product, people, and methods. SLCSE achieves a moderate level of project integration, enforcement of definitions, and definitional support [RUD89].

SLCSE supports DoD-STD-2167A by identifying all the data that is required for each development phase. Tools populate the database in the course of development. An automatic documentation generation tool can be used to extract information from the database to produce the required reports. The SLCSE roles restrict users to the data and tools required for a particular DoD-STD-2167A activity.

A design goal of SLCSE is to provide a consistent user interface across a range of terminal types and workstations. The SLCSE user interface is data driven and linked via a message handler to isolate the user interface from changes. For example, the introduction of a new tool into SLCSE is handled by specifying the tool name, its parameters, and the roles that are allowed to use the tool. This approach provides an easy facility for integrating a new tool into the SLCSE framework. SLCSE supports three levels of tool integration with its user interface:

1. The user interface libraries can be compiled directly into a tool to provide a SLCSE conforming look and feel to the tool. A number of system tools, such as *mail, dir, copy, delete,* conform to the SLCSE interface. A tool writer has the option of developing new tools to adhere to the SLCSE interface.
2. SLCSE provides a surrogate user interface process that supports a standard user interface protocol for SLCSE tools. Tools that use this protocol are isolated from host and terminal dependencies.
3. Tools imported into the SLCSE environment are simply invoked by the SLCSE user interface and will display their existing user interface.

SLCSE provides two kinds of transaction bindings. The first approach allows a user to request an exclusive lock on a given entity or relationship type, after which all instances of the object or relationship type participate in the transaction. The second alternative is similar to the traditional, single level, transaction mechanism in which all instances that are touched during the transaction participate in the transaction.

The SLCSE top-level architecture consists of three computer software subsystems:

- User Interface Subsystem
- Database Subsystem
- Toolset Subsystem

The Database Subsystem serves not only as a repository of software and project information produced by individual tools within the SLCSE Toolset, but also as a medium for information exchange between individual tools within the Toolset. The Database Subsystem provides the following functional capabilities:

- Schema Definition Language and Compiler
- SMARTSTAR
- Entity-Relationship (directly-coupled tool) Interface (ERIF)
- DCL (indirectly-couped tool) Interface

The SLCSE Database Subsystem is constructed on top of SMARTSTAR/IDM, a commercial relational database management system (DBMS). SMARTSTAR is a software implementation of a relational database which interprets Structured Query Language (SQL) and uses DEC's RDB database manager as its underlying database engine. SMARTSTAR/IDM is a SMARTSTAR look-alike and supports a hardware implementation of a commercial relational

database using the Britton-Lee IDM-500 Database Machine as it underlying database engine.

The SLCSE Database Subsystem has two distinct phases:

1) Creation
2) Access

During the **Creation** phase, Schema Definition Language (SDL) is processed by the SDL Compiler to generate a set of SQL statements that, when interpreted by SMARTSTAR, create the relational database tables which implement the ER model of the SLCSE Project Database. The SDL Compiler also generates the Schema Definition File (SDF), a runtime symbol table which is used by the EF Interface to allow directly and indirectly-coupled tools to reference database entities, relationships, and attributes by name.

During the **Access** phase, directly-coupled tools access SLCSE database entities, relationships, and attributes via the ER Interface (ERIF). The ERIF is an Ada package providing utilities which interpret ER queries, translate them into SQL statements, forward the SQL statements to SMARTSTAR, and process and return the results to the calling tool. Indirectly-coupled tools must utilize the DCL Interface in order to access the database. The DCL Interface consists of a set of directly-coupled utility programs. These utility programs process intermediate files containing database information in a standardized format which is translated into appropriate ERIF operations. The Schema Definition File (SDF) allows directly and indirectly-coupled tools to reference database entities, relationships, and attributes by name [STA88].

The SLCSE framework provides several features: an entity-relationship project database; common, menu-based user interface; simple transaction management; and user management. SLCSE is at a lower level of tool support for functionality than CAIS and PCTE [RUD89]. The higher level of tool support makes for a higher degree of tool integration.

5 Evaluation of Tool Interface Standards

The discussion of Tool Interface Standards is categorized into the following areas:

- Tool standardization efforts
- Tool interoperability
- Data exchange formats
- Language representations
- Documentation Representations

5.1 Tool Standardization Efforts by CFI

The Computer Aided Design Framework Initiative (CFI) is a consortium of over 50 corporations and institutions developing worldwide industry guidelines for electronic design automation frameworks and tools that remove the barriers to integration. Corporate members include AT&T, Cadence, DEC GM/Delco, HP, Honeywell, InterACT, MCC, Mentor, Motorola, National Semiconductor, NCR, NEC, Object Design, Objectivity, Sun TI, Valid Logic Systems, and VIEWlogic.

The CFI has defined guidelines for design automation frameworks to enable the coexistence and cooperation of a variety of tools [BUC90]. The CFI is standardizing the interface between CAD programs in order to reduce the barriers to tool/system integration. The CFI has exhibited a significant demonstration of 30 tools from 20 CAD vendors on a mix of 5 workstations running together through a CFI-defined procedural interface (PI). Most vendors are either already a part of the CFI movement, have plans to join the effort, or are at least keeping a close watch on CFI's developments.

5.2 Tool Interoperability

5.2.1 CEF

The Common Exchange Format (CEF) is a format of data exchange which is a subset of the Object Oriented Design Language (OODL). CEF objects are formed by adding a special syntax to OODL objects. The CEF objects may embody VHDL, EDIF, or other types of data streams, and instantiated design or administrative data objects. The CEF is being utilized in the Engineering Information Systems (EIS) effort by Honeywell, funded by the Air Force Wright Aeronautical Laboratories.

In the EIS design, the exchange of data is accomplished by the transfer of EIS-controlled objects among EIS system components by data exchange services. These services can transfer any number and any type of CEF objects in a single atomic operation. The atomic operation, performed in the foreground or background, allows the client to see either all or none of the effects produced by the data exchange service. A background operation has the capability of immediate or deferred initiation of the data transfer [HON89].

As part of the data exchange services in EIS, the sending client can specify the transfer of his data through a central integrated data repository for administrative or technical reasons. The client may also specify the transfer of any design data objects along with associated administrative information. For CEF transferred data, the data exchange service must automatically include administrative information and relevant information related to the transferred design data objects, origin and destination of the transfer, current state of the transferred object, or any configuration membership of the object [HON89].

At times, the receiving client may have to disassemble the exchanged data into its constituents. For CEF transferred data, it may be necessary to separate the administrative data from the design data objects. There must be some mapping to the engineering information model for EIS in order to identify what type of data the receiver has obtained from the sender. Thus, only the model-defined contents can be transmitted with any confidence [HON89].

5.2.2 Link Database

Using a Link Database is an approach to database integration and is part of the tool interoperability required in a tool interface [RUD89]. Database integration implies that tools share data and a description of the data. It also provides a uniform mechanism for managing and accessing the data so that the data is readily available, correct, and protected from intentional or unintentional access.

There are several approaches to linking the databases for tool interoperability in the tool interface:

1. **Common Database Interface Set:** A strategy for information integration involves the use of a common set of data management services interfaces. This approach is widely recognized as an effective means for integrating tools within an environment. A critical factor for the success of this approach is the adoption of a standard interface set and its acceptance by software tool vendors. CAIS and the PCTE are examples of frameworks that provide a set of interfaces for data management services.

2. **Common Data Dictionary and Multiple Databases:** The level of data integration can be enhanced by using a common data dictionary to provide a common understanding of the structure of the data, even though the data itself may be stored in physically distinct databases. It is common practice to develop data processing applications that access multiple databases through the use of a common data dictionary. The data dictionary becomes a single point of reference for all the project data definitions. A good example of this approach is the Information Resource Dictionary System (IRDS) being developed by the National Institute of Science and Technology. The IRDS supports the design of efficient programs and databases that share data. The IRDS uses a meta-data representation based on the Entity-Relationship-Attribute data model.

3. **External Links Mechanism:** Current environment framework databases do not provide adequate performance for kinds of data managed by CASE tools (e.g. intermediate forms). Consequently, most CASE vendors use a highly optimized and proprietary, application-specific data management system to support their tools. One problem with integrating these tools with other tools is the inability to establish traceability between information maintained by different systems. One strategy for dealing with this problem is to provide an external database that has a simple tool-independent interface for registering data dependencies.

The SUN NSE provides a linking service that allows tools which conform to the link interface protocols to establish relationships between data stored in their respective databases without having to directly access external databases.

The traditional issues of database integration still exist, such as, the data model, redundancy, replication, consistency, integrity, security, distribution, recovery, archiving, concurrency control, database/schema migration. The

engineering database issues are the engineering data, long duration, transitions, inconsistency tolerated, nested transactions, performance, and versioning. These issues are discussed below.

A **Data Model** must be supported by a software engineering framework and be sufficiently expressive to represent the complex data structures that occur in engineering projects. Both the CAIS and PCTE employ variants of the entity-relationship-attribute model. Other environments are experimenting with the object-oriented data models [WOL88]. An evolution towards more powerful data models will allow project data and project knowledge to be maintained within the project database. Increased power in the data model requires increased sophistication in the database management system as well as increased computing resources. The primary drawback to employing project databases based on these higher level models is that they are emerging technologies and will require several years to mature.

Data Redundancy, storing duplicate data, results in system inefficiencies and inconsistencies when a single change needs to be reflected in several places. Data duplication occurs in a software engineering environment for many reasons. A user may want a copy of some data with the intention of modifying the data. If the original data item needs to be updated, it is possible that copies of the data may also need to be updated. A framework should provide mechanisms that support data redundancy.

Data replication maintains the logical appearance of a single data item with an implementation that can make copies of the data item to improve data availability and survivability. Data replication requires that data locking be handled in a distributed fashion to ensure consistency of data values when updates are made. In addition, updates need to be propagated to all replicated instances of a data item before one is permitted to access an updated value. Supporting data replication reduces network performance and increases system complexity.

Data consistency implies that the data values within the database conform to the rules about their values. Transaction mechanisms are used to take a database from one globally consistent state to another. In an engineering database, this definition of consistency is not entirely appropriate. A typical engineering database may not achieve a consistent state for weeks or months, and in some cases may never reach a globally consistent state. The database should allow the user to abort an incomplete transaction and undo or redo completed ones as a way of backtracking under the user's control.

Data integrity specifications describe the rules under which a data item value is well formed. Integrity checking includes conformance to data typing or

dynamic triggering of constraint checking routines. Each of these mechanisms introduces a performance overhead for making changes to framework data.

The Framework **Security** model may need to support user authentication, discretionary and mandatory access control, and encryption, depending on the project requirements, the framework security. Security mechanisms become increasingly more complicated as the data model becomes richer. In addition, not all data within a project will require the same level of access control. All of these mechanisms impose an overhead. However, if higher levels of security are required, then these mechanisms need to be considered. To date, most of these mechanisms have yet to be applied to software development frameworks.

Distribution is the ability of a network node to maintain global knowledge of the total network in order to make local decisions. The ability to maintain global knowledge requires significant network communication overhead and delay. Data distribution includes the allocation of data to nodes in a network, location transparent access to data, replication of data to improve availability and survivability, decomposition of database queries into subqueries that can be sent to different nodes, management of locks and names on a network wide basis, check pointing, recovery, and initializing of nodes after a failure. These mechanisms introduce complexity and have significant performance implications.

Recovery is the activity of ensuring that the framework can restore itself to some previous consistent state after a failure. An environment framework must provide tools and procedures to automate system recovery after a failure. Manual recovery procedures will be inadequate as the data complexity managed by the framework increases.

Archiving and restoring data is a difficult problem when the framework supports higher level data models. It becomes more difficult to determine the set of objects and their relationships that need to be saved. Furthermore, the restoration process must insert both objects and relationships into an existing structure. Both of these problems are not fully understood at this time.

Concurrency control is the activity of coordinating the actions of processes that concurrently access shared data and could potentially interfere with each other. The simple models of concurrency control employed in file based systems may prove to be totally inadequate when higher level data models are employed. When a large number of objects are accessed by an operation, there can be a substantial performance overhead in implementing

concurrency control (especially in a distributed environment) and in knowing exactly what must be locked.

An environment framework must support **Database Migration,** dynamic changes to the data definition describing the structure of entities, relationships, and behavior of data without having to take the system off-line. A change to a data definition can entail changes in storage allocation for entities, adjusting existing data values and generating new ones, and restructuring relationships among objects within the database. The data management facilities within the framework must provide tools to support these activities and ensure that the data migration activities result in a consistent database.

Engineering applications require that the database supports the manipulation of **Engineering Data,** complex objects and storage of graphic, binary, and textual data of varying sizes. Most existing commercial databases support a limited set of data types and do not handle large and varying size data types.

Versioning, successively refining information about a system is required by engineering data. The database must support versioning so that different temporal states of an object can be referenced to recreated an earlier state of the information about a system. One issue with versioning involves deciding which versioning model is appropriate. The strategy of immutable objects is gaining acceptance because it eliminates some difficult problems of maintaining database consistency. The notion of immutable objects is that once an object is committed, its values are time invariant. The versioning model is intimately related to the database concurrency control and transaction mechanisms.

As an assessment of link databases, information integration is critical for providing effective automation within an environment. Current framework standards efforts and emerging commercial frameworks are addressing the issue of providing standard database interfaces. The PCTE is the most advanced commercially available framework that supports a distributed entity-relationship project database. At the current level of PCTE funding and the high level of industry participation, the PCTE will continue to evolve and mature rapidly. The PCTE is commercially available on most of the common computer platforms. PCTE may serve to help address the issues of link databases, tool interoperability, and ultimately tool interfaces.

5.2.3 Clipboard

The systems engineering environment should provide both private and public clipboard mechanisms. Private clipboards allow data to be transferred

between two tools executing simultaneously. A clipboard is an approach to information interchange [RUD89].

Clipboard facilities can be used as temporary buffers to allow information presented by one tool to be transferred into another tool using a simple cut and paste operation. The Apple Macintosh™ illustrates the power of a simple clipboard mechanism that supports standard graphic and text data. Graphics and text can be cut from any application and pasted into another.

Issues that need to be addressed in the use of a clipboard as an information interchange mechanism are the proliferation of exchange formats, the lack of standards, the loss of information, the reduced usability, and the lack of automation [RUD89].

5.2.4 Pipes

A pipe is a Unix facility for redirecting text I/O. In systems such as UNIX, a virtual file joins two processes together, effectively 'connecting' two commands together. This connection, known as a pipe, enables the user to take the output from one command and feed it directly into the input of another command. Pipes can be thought as a measure of tool integration [TER90]. A pipe is effected by the character '|' which is placed between the two commands [KOC84].

When a pipe is set up between two commands, the standard output from the first command is connected directly to the standard input of the second command. A pipe can be made between any two programs as well as any two commands under the UNIX system. It is also possible to form a pipe consisting of several programs, as a double pipe.

It is important to note that the size of a pipe is finite. In other words, only a certain number of bytes can remain in the pipe without being read. The limit is typically 5120 bytes. Although generous enough for most purposes, this maximum size has implications for both write and read commands. If a write is made that will overfill a pipe, process execution is normally suspended until room is made by another process reading from the pipe.

Pipes support tool composition, provided the tools operate on text files. It is the user's responsibility to ensure that the specified composition is meaningful.

5.2.5 Object Request Broker

The OMG (Object Management Group) issued a Request for Proposal (RFP) with submissions due in March 1991, to begin the process of filling the open portions of the OMA. The Technical Committee (TC) will make recommendations to the OMG Board in July 1991. Central to the OMA is the Object Request Broker (ORB) which provides the mechanism by which objects transparently make and receive requests and responses. The ORB further provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems [OMG90].

A request is an event, (i.e., something that occurs at a particular time during the execution of the computational system. The information associated with a request consists of an operation and zero or more (actual) parameters. Operations are potentially generic, (i.e., a single operation can be requested of an object with different implementations, resulting in observably different behavior. A request causes a service to be performed on behalf of the client. One outcome of performing a service may be that some results are returned to the client. The results associated with a request may include values as well as status information indicating that exceptional conditions were raised in attempting to perform the requested service [SOL90].

The behavior of a request is the observable effects resulting form performing the requested service. The effects may be visible to parties other than the requesting client. The behavior of a request includes the results returned to the client including both the values returned and the exceptional conditions reported, as well as indirect effects on the results of future requests by the same or different client [SOL90].

The practical effect of performing a requested service is to cause some code to execute that accesses some stored data. The stored data represents a component of the state of the computational system. The code performs the requested service, which may change the state of the system. The code that is executed to perform a service is called a method. An OMG system includes an infrastructure that serves as the mediator between clients and services. A primary function of the infrastructure is to select the appropriate code to perform requested service, and to execute that code giving it access to the appropriate data [SOL90].

The selection of a method to perform a requested service and the selection of the data to be accessed by the method is called a binding. A binding can be

static, the selection is performed prior to the actual issuing of the request, or dynamic, the selection is performed after the request is issued [SOL90].

An application which is compliant with the Object Management Architecture (OMA), as suggested by the Object Management Group (OMG), consists of a set of interworking classes and instances which interact by the Object Request Broker. Figure 5.2.5-1 shows the four major parts of the OMG's Object Management Architecture and its relationship to the Object Request Broker [SOL90]. The solid icons represent software that have application programming interfaces. The dotted icons represent groupings of objects that can make and receive requests.

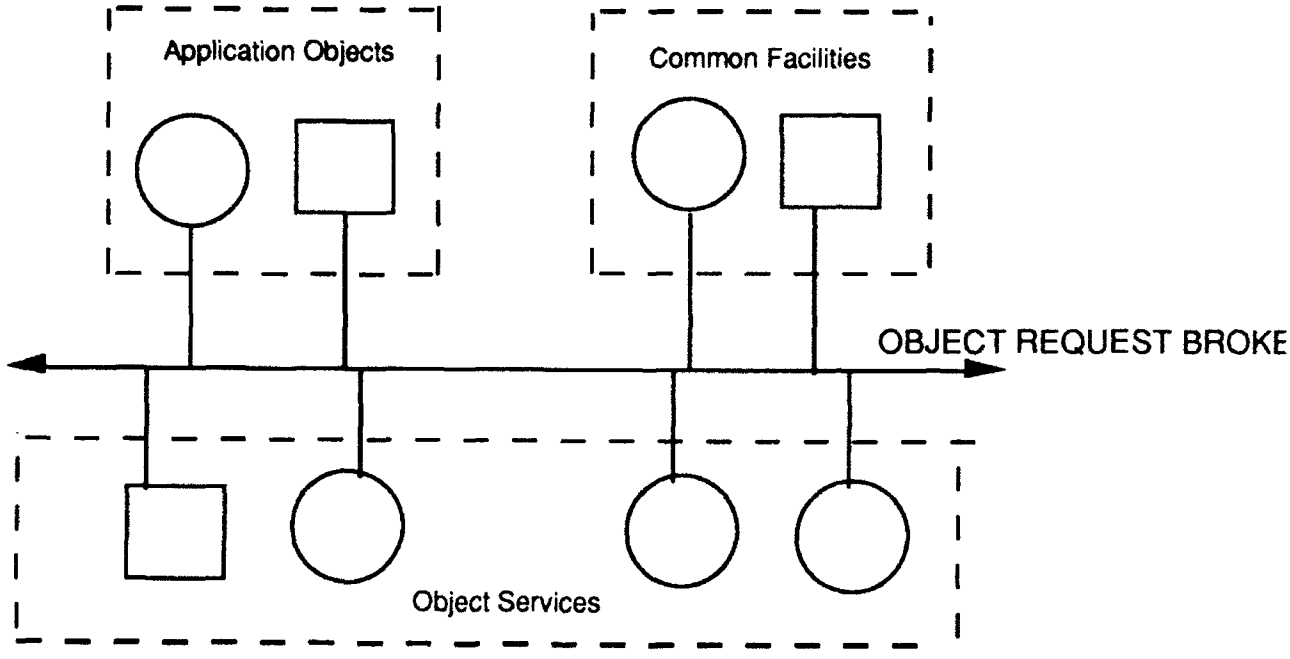


Figure 5.2.5-1 Object Management Architecture Overview [SOL90]

The components in the Figure 5.2.5-1 are described as follows [SOL90]:

- The Object Request Broker (ORB) enables object to make and receive requests and responses.
- Object Services (OS) is a collection of services with object interfaces that provide basic functions for realizing and maintaining objects.
- Common Facilities (CF) is a collection of classes and objects that provide general purpose capabilities that are useful in many applications.
- Application Object (AO) are specific to particular end-user applications.

The OS, CF, and AO define a partitioning in terms of function and will be the focus of the OMG standardization efforts. In general, the OS CF, and AO all intercommunicate using the Object Request Broker. Existing application, external tools, and system support software can be embedded as object that participate in the Object Management Architecture, using class interface front-ends, commonly called adapters or wrappers [SOL90].

The ORB itself might not maintain all of the information needed to carry out its functions. In the process of conveying a request, the ORB may generate requests of its own. For example, in order to find the specific method to be executed for a given request, the ORB might use a class dictionary service or might search run-time method libraries [SOL90].

The ORB is expected to address all of the following areas, at least to some degree [SOL90]:

- Name services. Object name mapping services map object names in the naming domain of the requestor into equivalent names in the domain of the method to be executed, and vice versa. The OMG Object Model does not require object names to be unique, nor universal.
- Request dispatch. This function determines which method to invoke. The OMG Object Model does not require a request to be delivered to any particular object.
- Parameter encoding. These facilities convey the local representation of parameter values in the requestor's environment to equivalent representations in the recipient's environment. To accomplish this, parameter encoding may employ standards or de facto standards (NFS, NCA, ASN.1, etc.).
- Delivery. Requests and results must be delivered to the proper location as characterized by a particular node, address, space, thread, entry point. These facilities may use standard transport protocols (e.g., TCP/UDP/IP, ISO/TP).
- Synchronization. Synchronization primarily deals with handling the parallelism of the objects making and processing a request and the rendezvousing of the requestor with the response to the request.
- Activation. Activation is the housekeeping processing necessary before a method can be invoked. Activation and deactivation of persistent objects is needed to obtain the object state for use when the object is accessed, and save the state when it no longer needs to be accessed.

- Exception handling. Failures and their recovery, in the process of object location and attempted request delivery will occur and those must be reported to requestor and/or recipient in ways that distinguish them from other errors.
- Security mechanisms. The ORB provides security enforcement mechanisms that support higher-level security control and policies.

To illustrate the function of the ORB, consider the request "print layout_312 laser_plotter". This could be sent to the object "layout_312" whose "print" method would then print it on "laser_plotter". Or the request could be sent to "laser_plotter" whose "print"method would access "layout_312". Or the request could be sent to a generalized "print"routine that would determine the best way to arrange the printing, based on some attributes of these two object. Or, instead of relying on a generalized "print"routine, the Name Service in the ORB could determine an appropriate method jointly owned by the classes of "layout_312" and "laser_plotter".

The specifications for the ORB in the RFPs must conform to standards, be portable, commercially available, and be fully documented. The submissions must address several key questions with answers to integrations with foreign object systems, binding, concurrency, representation, granularity of objects, extensibility, internationalization, use of technology, object lifetime, and limitations [OMG90].

5.3 Data Interchange Formats

5.3.1 PDES/STEP

The Product Data Exchange Specification (PDES) has a history that has evolved quickly. In 1988, the CAD data transfer community, issued the IGES (Initial Graphics Exchange Specification) in a new version (4.0). The IGES was one of the main CAD-CAM standards, but it grew out-of-date, and its limitations were becoming more difficult to deal with. An international industrial consortium, PDES, Inc., was formed in the summer of 1988, to help implement PDES-STEP (Product Data Exchange Specification - Standard for the Exchange of Product Model Data) ISO TC 184/SC4. This industry consortium formed to develop a technical, product data exchange database to facilitate a more efficacious use of digital representation standards. A product of this consortium was the new standard, PDES, to eventually replace IGES. Organized meetings of this newly formed consortium just started in 1989, followed by another meeting in July 1990. Other standards exist, usually for particular industries or applications, and must be incorporated into PDES [FAU88].

Used with PDES, the Specification for Exchange for Product Model Data (STEP) was voted to the ISO draft proposal stage. STEP will enable users with different computers to contribute, access and share mechanical, electrical and structural information not previously available in a standard format. The PDES-STEP Integrated Product Information Model is intended to be hardware independent. Four levels of implementation are proposed: file exchange, working-form exchange, database exchange, and knowledge base exchange. As of 1988, the IGES-PDES and ISO communities have concentrated on level 1 (file exchange) implementation.

Sharing information between CAD systems can be done in one of two ways: using a neutral or base-line language such as IGES or PDES or translating directly between each of the languages. The main problem with a neutral language is the huge variety of proprietary CAD data structures it must handle. Converters of direct database converts eliminate this problem by translating from the source system to the target system in a direct, single-step process. Other advantages of direct database converters include running the conversion on a single host system; faster conversion; formatting output data in the customer's own CAD-CAM system; and are cheaper to purchase and operate.

Product Data Exchange Specification (PDES) is an evolving specification that will be accepted by the American National Standards Institute as a standard for defining and exchanging intelligent object data among computer-aided-design (CAD) systems. PDES contains the functionality of the Initial Graphics Exchange Specification 3.0 and inputs to the international standard for the Exchange of Product (STEP) model data that will be implemented as a standard by the International Standards Organization.

The use of PDES as a common database is the mission and objective for the DoD-based effort of the Computer Aided Acquisition and Logistics Support Software Products Committee (CALS). CALS is a paper reduction initiative mandating the electronic delivery of engineering documentation and storing it in a common database. CALS will use the developed document type definitions using ISO 8879 SGML for the Data Item Descriptions (DiDs) related to DoD-STD 2167A and DoD-STD 7935A. CALS also has identified STEP, PDES' international cousin, as a standard for the exchange of product model data. These standards will act as guides for all the information produced in the design and support of weapon systems.

PDES defines a neutral object data format for architecture, engineering, construction, mechanical, and electrical CAD to enable sharing of CAD files between disparate CAD systems. The standard includes three layers:

conceptual information modeling, integration of application and topical models, and a physical file definition.

PDES has centered on product descriptions for mechanical design. PDES will extend activities into the electrical domain, working with national standards organization (e.g., IEEE, EIA, etc.). The various standards development organizations for PDES product information will express the components of PDES in a language conforming to existing standards efforts, but then will convert or relate semantic components to a common integrated information model using the Express language. The PDES voluntary is a group supporting the development of PDES electrical in the Electrical Application Committee, using the PDES Information Modeling Methodology.

ANSI PDES Electrical Engineering standards are now under the guidance of an organization with a board consisting of EIA, IEEE, Institute for Interconnectivity and Packaging Electronic Circuits (IPC), and the American Society of Mechanical Engineers (ASME). Task groups will typically work the issues of the information model language, the integrated model, implementation, validation and test, and the standards taxonomy.

PDES, Inc. is providing industrial validation of PDES standards. The national PDES electrical effort will be guided by subcommittees, reporting to ANSI. NIST is providing a PDES testbed. At present, there are discussions regarding the integration of an EIS prototype into the NIST PDES testbed. The EIS/ECAD model extensions also should be integrated with the PDES models. This effort could yield a multidomain demonstration.

PDES is being touted as the data exchange technology of the 1990s [WAR90]. A coalition of vendors, users, educators, and government representatives are working to implement PDES as an international standard by 1991. PDES is an outgrowth of the ten-year-old IGES standard, which covers graphics and geometric data. It goes beyond the old technology by adding the ability to exchange products model and application data. PDES has the flexibility to provide a platform for both static and dynamic data exchange for users in three CAD areas: mechanical engineering, electrical engineering, plant design, construction models, road and bridge construction and shipbuilding. The standard utilizes a three-layer approach to applications development, moving from conceptual model sot integrate models to physical file definition. PDES Version 1.0 is expected to debut in 1990. Colin McMillan, assistant secretary of Defense for production and logistics, said as STEP becomes available, it will be a requirement in DoD contracts [STA90]. Commerce and the DoD will expand PDES research at the National PDES Testbed at the National Institute of Standards and Technology, an agency of the Commerce Department.

5.3.2 CDIF

CASE Data Interchange Format (CDIF) is a EDIF 300 (Electronic Design Interchange Format) specification of data interchange among CASE tools developed by EIA (Electronics Industry Association) [TER90]. CDIF, as well as EDIF (Electronic Design Interchange Format), are standardized semantic level portability services.

The EIA's objective is to develop an ANSI standard for the data exchange between CASE tools. The EIA is a voting member of ANSI. The CDIF Technical Committee will identify areas to automate, construct models to represent these areas, and provide a mapping from the models to the transfer syntax. Currently, EIA is develop a draft standard for 'upper CASE' data exchange between similar tools in the same life cycle phase which includes a meta-meta model describing their modeling technique. EIA's first release for public comment is scheduled for late in 1990. followed by public ballot by mid 1991.

EDIF is a comprehensive semiconductor data exchange format that facilitates the transfer of schematic and symbol libraries between vendors, manufacturers, and customers. EDIF has its own semantics and is fully expressible in object format [ROL90]. By greatly facilitating faithful exchanges of designs, EDIF spans proprietary boundaries and protects investments made in diverse design environments. EDIF Version 200 (EIA-548) was adopted as an American National Standard on March 14, 1988. EDIF Version 200 is the upward compatible kernel of any future versions of EDIF. EDIF 200 is a widely-adopted, broad-spectrum, vendor-neutral standard that expresses netlist, schematic, and layout design data [ST90d].

These two services, CDIF and EDIF, are currently being supported under the EIS (Electronics Information System) project. The CASE and information requirement (e.g., IDEF-like) standards are good for expressing system requirements. Transfer of data, models, and final designs demand mutual agreement between standards organization and component vendors. Although the CFI will provide low-level interchange, higher order semantics are possible using other protostandards. EDIF and CDIF are two EIA-sponsored standardizing bodies developing standards for interchange of electronic circuit and software engineering (CASE) tool information. The latter effort is also being considered by the IEEE Computer Society Task Force on Professional Computing Tools [ROL90].

5.3.3 ODA/ODIF

Office Document Architecture/Office Document Interchange Format (ODA/ODIF) is a standard for the structure of documents for interchange between cooperating tools in office systems. The International Telegraph and Telephone Consultative Committee (CCITT), the International Organization for Standardization (ISO), and the European Computer Manufacturers Association (ECMA) have all been, or still are, working on the standards, and the results obtained by ISO and ECMA are essentially the same for both organizations. CCITT Group 4 raster graphics for storing and retrieving engineering documents has been identified as a standard for CALS.

Office Document Architecture (ODA) is a document architecture for document communication between document processing tools (editors, formatters) in multivendor office systems. It is a modern, object-oriented document architecture for the description of both the logical and layout aspects of a document. Parts of the ODA have similarities to the 'metaclasses' of object-orient programming languages [CRO87]. ODA contains means to allow the application to define and interchange document class descriptions with construction rules, content rules, and layout directives to facilitate document preparation. ODA further allows for application-defined attributes.

ODA establishes a document architecture model which identifies definitions for terms used in the office system. For example, a document is a structured amount of text that can be interchanged as a unit between an originator and a recipient. A document can be interchanged either in image form, to permit its being printed and displayed as intended by the originator, or in processible form, to permit document editing and layout revision by the recipient.

Text is defined as a representation of information for human perception that can be reproduced in two-dimensional form. Text consists of graphic elements such as graphic characters, geometric elements, photographic elements, and combinations of these. Text and additional control information constitute the content of a document.

Any given document is characterized by its content and its internal organization. ODA distinguishes between a document's logical structure and its layout structure. The logical structure associates the content of the document with a hierarchy of logical objects, whereas the layout structure associates the same content with a hierarchy of layout objects. Examples of logical objects are summaries, titles, sections, paragraphs, figures, tables, etc. Examples of layout objects are pages, columns, and areas with contents of different categories.

Before a document can be imaged, its layout structure must be established. The graphic elements of the content associated with the logical objects must be arranged within certain layout areas that represent the layout objects. During imaging, these areas are then mapped onto a physical presentation medium.

Objects have properties such as dimensions and, in the case of layout object a position, and relationships exist between objects. Properties and relationships are expressed by attributes assigned to the objects.

ODA's architectural model distinguishes between composite objects and basic objects. Composite object consist of components that may be other composite objects and/or basic objects. Basic objects are at the lowest hierarchical levels, and it is only through them that content portions are directly associated by means of the attribute references to content portions.

In the interchanged data stream, each content portion is represented by a set of attributes and a string of graphic elements with control elements. Each content portion is identified in the data stream by the attribute content portion identifier. Each content portion is structured and presented according to a specific content architecture depending on the category of their graphic elements. A content architecture defines presentation attributes of basic objects as well as the set of graphic elements and control functions with their coded representation. ODA currently defines a character content architecture and a photographic content architecture.

The Office Document Interchange Format (ODIF) defines the general structure of the data stream as a sequence of descriptors and text units. A descriptor is a composite data item consisting of subdata items and basic data items representing the attributes of a document profile, an object definition, or an object. A text unit represents a content portion. It is a composite data item consisting of subdata items and basic data items that represent the attributes of the content portion, and a set of one or more basic data items that represent the graphic elements of the content portion.

The formal specification contained in the ODA standard is based on the Abstract Syntax Notation ASN.1 defined in ISO 8824, which is essentially the same as the Standard Notation of the Presentation Transfer Syntax specified in CCITT Recommendation X.409/12. In this syntax, each item of information is viewed as a data item of specific data type with a specific data value. The syntax notation is used to define the format of the data flow and its components as a sequence or set of more elementary data types that are, in

turn, defined by means of more elementary data types; this nested specification results in the basic data types as integer and octet string.

Coding must be performed according to the encoding rules for the abstract syntax notation ASN.1 defined in ISO 8825. Coding renders the document as a sequence of octets.

Besides additional content types, future extension should include [HOR85]:

- Default value lists for type-and class-specific attributes
- Object identifier expressions that allow objects of a certain class to be accessed
- Background color for layout objects and character boxes
- Support for a more sophisticated layout by means of flexible frames
- Support for automated treatment of cross-references
- Registered font layouts
- A syntax for transformation rules to generate business graphics from data in spreadsheets
- Enhanced construction rules to support arrays, e.g. tables
- Property rules that are not restricted to constant expressions
- Support of external references to parts of documents and to document class definitions already available to the receiver
- Interchange formats for parts of documents for distributed documents preparation
- Support for automatic generation of tables of content and of indexes
- Support of gray-level images and half-tome image
- Security attributes in the document profile

5.3.4 TIFF

Tagged Image File Format (TIFF), produced by Aldus/Microsoft, is the most flexible and reliable method for storing bit-mapped images with various sizes, resolutions (number of dots per inch), number of grays, and colors. TIFF provides a common format for scanners and desktop publishing software [GRA89]. TIFF was created specifically for storing gray-scale data, and is the standard format for scanned images such as photographs.

TIFF has three subtypes that are distinguished by the number of colors or gray shades they contain [PAR90] depending upon the software that creates the image:

- Monochrome TIFF stores only 1-bit images, but the black and white pixels can be dithered in a variety of patterns to simulate grays quite accurately. On the other hand, dithered patterns limit the degree to which the user can edit and scale an image.
- Gray-scale TIFF typically holds 256 grays. It is the best choice for graphics used in page layouts, because most page-design programs can adjust the contrast and brightness of a gray-scale TIFF image.
- Color TIFF can hold up to 16.8 million colors. It is the preferred format for image that will be color-separated.

TIFF helps resolve the problem of many systems remaining graphically isolated because of the lack of uniformly accepted standards for storing and transferring graphics. However, it cannot store object-oriented images [PAR90].

Although TIFF is considered a standard graphics format, some programs save TIFF images with subtle variations from the norm. Although TIFF was intended to provide a high-resolution bit-mapped standard, users wanted to improve it–and did, in different ways–so the TIFF standard isn't exactly standard anymore [AKE87]. These users variations may prevent the files from being opened by other applications. Note, there is the similar-sounding RIFF (Raster Image File Format), which is not a variation of TIFF, but rather a compressed format used by ImageStudio and ColorStudio; it can be imported by several other applications [PAR90].

Most graphics formats share some common elements. The shape-defined format that defines an image as a series of geometric shapes and patterns is one of the most simple and versatile graphics formats, the ANSI X3.110-1983 North American Presentation-Level Protocol Syntax (NAPLPS) Standard. Many computer-specific formats have emerged that utilize the most efficient storage systems available on a machine, but make it difficult to transfer graphics between machines. The Graphics Interchange Format (GIF) for color-image transfer supports image dimensions up to 64,000 pixels, multiple images, rapid on-line decoding for viewing, and hardware independence [GRA89]. These characteristics may serve to be a basis for standards to circumvent the idiosyncrasies presented in data interchange formats.

TIFF is the de facto industry standard format for storing bit-mapped images and is now found in most desktop publishing and scanner software for the

Macintosh and PC systems. TIFF currently has a Library Package in public domain. TIFF's success in the industry stems from its capability to store image details in a superset of various existing proprietary formats [MEA88].

However, Mark Zimmer, the developer of the ImageStudio, a gray-level editing program, says that buyers must beware when purchasing a image scanner software. Scanner users will be frustrated by the proprietary approaches taken by various companies until a standards committee is formed for graphics file formats used in desktop publishing and image scanning software, or until one file format is recognized as the de facto standard. For example, the TIFF is Aldus Corporation's PageMaker package is too complex and inadequate for storing compressed images, according to Zimmer [BAN88]. Datacopy VP Jim McNaul claims the lack of a standard file format is more an annoyance than a serious problem, but he says users must keep the different formats in mind when purchasing a scanner.

5.3.5 PHIGS

The Programmer's Hierarchical Interactive Graphics (PHIGS) standard defines a sophisticated graphics systems that controls the definitions, modification, and display of hierarchical graphic data. With numerous CASE tools and drawing tools, there is often a need to export or import graphic drawings that can be manipulated further for presentation purposes, and PHIGS is designed to meet this need. PHIGS supports coupling between applications programs and graphics systems, and specifies functional descriptions of system capabilities, including the definition of internal data structures, graphics editing capabilities, display operations, and workstation control functions [RUD89]. PHIGS supports hierarchical structuring of graphical data, interactivity, and PHIGS may be applied to distributed environments [CAH84].

The graphics communities requiring a graphics support system provided by PHIGS are Computer Aided Engineering, Process Control, and Command & Control [PHI84]. The benefits of a standard for these graphics communities are the following [PHI84]:

- Application programs that utilize PHIGS to be easily transported between installations.
- Functions currently performed by the application program are performed by PHIGS, and consequently, more time can be allocated to application function development rather than graphics support development.
- Equipment manufacturers can use PHIGS as a guideline for providing graphics device capabilities appropriate for the graphics community.

PHIGS is governed by a number of global philosophical positions [PHI84]:

- Multilevel/hierarchical structuring of graphics data
- A high degree of interactive computer graphics
- Rapid modification of graphics data and the relationships among the data
- Applicability to a diverse application family
- Minimum set of functions while producing an effective standard

PHIGS supports the description of a **multilevel/hierarchical structuring** between components of the application graphics data. The hierarchical description may be of arbitrary depth and may include data sharing among levels. A hierarchical data organization is selected over a single level organization due to its advantages in describing geometric relationships, its usefulness in simulating motion, and its effectiveness in performing display changes to a large number of named graphics objects [PHI84].

Interactive computer graphics is viewed as the principal use of PHIGS. PHIGS functionality encourages the use of graphical input devices in a highly interactive manner and will include capabilities valuable in an interactive environment [PHI84].

In support of an interactive environment, PHIGS supports **rapid modification of graphics data**. Limitations of the possible modifications to the data will avoided, improving support of application required techniques. The PHIGS graphics data organization will favor rapid interpretation and display of the graphics data [PHI84].

PHIGS is a suitable graphics standard for applications in **diverse application** disciplines. Application-specific capabilities will be avoided. Specific functions reflecting a single discipline's common practices and requirements may be built atop the general purpose functions provided [PHI84].

The PHIGS standard is composed of a **minimum set of functions** in order to encourage simplicity to make the standard development an achievable task. In general, redundant functions are avoided; however, redundant functions are supported where their availability optimizes application use of the standard through frequently used capabilities or significant performance improvements [PHI84].

PHIGS is defined as a functional specification of the interface between an application program and its graphics support system. The major concepts of the functional specification of PHIGS involve the following areas [PHI84]:

- Environment
- Graphics data organization
- Input functionality
- Error detection and reporting
- Graphics data storage centralization/decentralization

PHIGS supports an **environment** of multiple logical graphics workstations. Each workstation is capable of either providing input, generating an image from the graphics data, or both. While the PHIGS workstation does not conceptually store the graphics data locally, it does maintain workstation-dependent attribute tables and has access to the centralized graphics data storage [PHI84].

The **graphics data organization** and the data storage is conceptually a unique, centralized facility independent of the existence, abilities, or status of any particular workstation(s). When the application specifies graphics data to the PHIGS system, it is entered into the graphics data storage.

In data storage, the graphics data is organized into units called 'structures.' Structures provide a mechanism for grouping the basic data elements. These elements, which are known as 'structure elements' include the traditional output primitives, attribute selections, viewing selections and modeling transformations, as well as labels, pick set elements, application specific data and execute structure elements.

The time at which graphics structures are displayed is independent of the time of entry into the data storage. The application may at any time identify a particular structure for display on a workstation, either before, during or after the structure data has been entered into the storage. When a structure is identified for display on a workstation, any elements in that structure as currently entered in the data storage are traversed and sent to the workstation. Any subsequent changes to the structure will be reflected on the workstation until the structure is identified for removal from the workstation. Removal of a structure from display does not affect its existence or definition in the data storage.

Graphics data in the storage can be changed in several ways. Additional structure elements can be added and existing elements can be accessed and deleted. Structure modifications are performed through structure editing. These operations are independent of the type of structure element. This functionality provides a flexible, rapid means of data modification in the data storage and on the display.

Because structures can contain elements which cause the execution (traversal) of other structures, an application may represent hierarchical relationships among its data. While structure content is limited with respect to data elements, the order and use of these elements within a structure is not restricted. This permits application flexibility in the representation of its data.

The attribute selection of structure elements in PHIGS apply hierarchically. The value of an attribute affects only the structure in which it is defined and subsequent structures below it in the hierarchy. This is in keeping with the goal of hierarchical structures by describing the logical relationships between those structures.

The application may also indicate to PHIGS that it wishes to enter data into a structure which need not be entered into the data storage. This is called the 'non-retained structure.' This non-retained structure is not modifiable. Like structures in the data storage, this structure may be identified for display on a workstation either before, during or after definition. However, a workstation will display only those output primitives specified by the application after this structure has been identified for display on that workstation.

The **input functionality** supports the common graphical and non-graphical input devices used in interactive graphics. This support is provided without mandating a certain application input methodology, but rather by providing the basic tools for application interaction. Input functionality will be used to control display data editing in an interactive application. For this reason, the

input functionality will provide sufficient information for subsequent modification of the graphics data by the application.

In support of **error detection and reporting** in the desired interactive environment, PHIGS is designed to be a system with few error states. This limits the amount of error checking and reporting required in a PHIGS implementation. In general, the existence of error conditions usually associated with graphics systems may be detected through the use of inquiry functions. The limited importance on the detection and reporting of errors is intended to provide increased performance in a PHIGS implementation.

The **centralized/de-centralized graphics data storage** in PHIGS implies that all graphics data storage is available to all workstations and no workstation-dependent structure storage is required. This conceptual model does not preclude an implementation which physically stores graphics data on workstations or any other way, provided that all the graphics data is available at all times as if it were located in a single central location.

Proponents of PHIGS believe that other standards, namely, the Graphical Kernel System (GKS), do not adequately satisfy the requirements of application programs. PHIGS provides functions that the GKS standard does not, such as support for three-dimensional graphics. PHIGS's functionality has been adapted, to a fair degree, from GKS and Core standards, however, but offers better performance in several areas. PHIGS goes beyond these other standards by providing definition, display, and modification of 2-D or 3-D graphical data, geometrically related objects, and rapid dynamic articulation of graphical entities [PHI84].

While PHIGS is intended to serve a different constituency than the GKS system, compatibility with GKS is a highly desired goal. The relationship of these two standards is important to the success of both of these standards in serving the graphics community. For example [PHI84],

- Technical differences will be allowed based on the needs of the PHIGS community and their objectives.
- If PHIGS and GKS share an identical function, the functional arguments, language binding, and effect of any attributes will be the same.
- Error states will be reduced in PHIGS where possible. Some GKS error states will not be retained, but may be check within a shell built atop PHIGS.
- Functionality beyond that provided in GKS will have, as its default, the GKS functionality whenever possible.

Extensions to PHIGS, known as PHIGS+, address the capabilities of the "ultrahigh-performance workstations," which can support interactively, in real time, such tasks as hierarchy traversal, geometric transformations, lighting, and shading of 3D data [CHI88]. The PHIGS+ standards define the surface rendering extensions to PHIGS (ANSI/ISO), and PEX supports PHIGS implemented under X, the PHIGS library and X Windows. PEX is an ANSI-based software standard for running, in a windowed environment and 3-D applications.

The proposed PHIGS standard is being reviewed for adoption as a standard by ANSI and ISO. Interactive graphics is provided by PHIGS using the ISO 9592 specification for the NIST-Application Portability Profile (APP). A product introduced by TGS, named Figaro, is the first major implementation of PHIGS on VAX/VMS, IBM VM/CMS, MVS/TSO, Apollo SUN, Silicon Graphics, HP, Masscomp, Prime, Steller, and Tektronix workstations [RUD89].

5.4 Language Representation

5.4.1 PostScript

PostScript was developed by Adobe Systems as an interpreted, stack-oriented language for describing, in a device independent fashion, the way in which pages can be composed of character, s shapes, and digitized images in black and white, gray scale, or color. PostScript is now the most widely used printer controller in the industry. Adobe is currently concentrating its development efforts on Display PostScript for computer graphics monitors.

PostScript is a text-based description language of an image to indicate how fonts and graphics are actually created on-screen. The beauty of pure PostScript is that the user does not need the originating program to print the file. The PostScript file can be fed directly to a PostScript printer with a simple PostScript download utility. Unfortunately, there is no preview image, and the graphic essentially loses its editability, so the user should always keep the original version of an image in the native format of its creating application [PAR90].

PostScript does not have the displayable PICT image that Encapsulated PostScript (EPS) offers. PostScript may be the best-know imaging model, familiar from laser printers. For example, the typeface and size of text in a word processor or desktop publishing program is specified through the imaging model, as well as the lines and curves of a CAD program.

PostScript and PostScript related products had a booming year in 1988. Hundreds of PostScript service bureaus sprung up, creating a new industry almost overnight. Color was the major issue, the first color PostScript printer was announced and started shipping, and several packages come out with color separation capabilities, most notably Illustrator 88 and PhotoMac. Several other programs with support for the QMS Color PostScript printer and color separations are on the brink of announcement or shipment.

The first clones started showing up on the market in 1988, with a lot more hype and broken promises than actual software and hardware. Once the clones start proving themselves and the initial fears in the PostScript marketplace die down, the clone business should turn into a rather large segment of the PostScript marketplace; However, Patrick Wood, editor of the *PostScript Language Journal*, feels that Adobe will continue to lead the PostScript market with new releases, features, and improvements [WOO89].

During the Seybold Conference in 1989, important announcements were made concerning Apple, Microsoft, and PostScript which may impact the industry, users and suppliers alike. Debates continue over the perceived impacts. What has been agreed upon between Apple and Microsoft and what is to happen can be summarized briefly as follows:

- Microsoft will license Apple's new Royal font format and software for scalable outline fonts to its OS/2 operation system and for the IBM PC and compatibles.
- Microsoft will develop its own clone version of PostScript for printers and will license it to Apple (who will use it in a future version of the LaserWriter) and to other printer manufacturers. It will be based on Adobe PostScript but will include extensions to make it more compatible with the graphics formats used in the PC and the Macintosh.
- Adobe will publish the details of its Type 1 font format, and the encryption and hinting mechanisms used, so that anyone can develop fonts in Adobe PostScript format, and so that PostScript clone printers can accept these fonts. Adobe also invites Bitstream (a major competitor in the supply of typefaces) to share in the openness of its Type 1 font format. Adobe will also publish the information in a form suitable for adoption by the international standards committees.

Microsoft' motivation to license Apple's new Royal font format and software for scalable outline fonts came from the missing ability to move a text document between machines and have the text look the same on the screen with identical line and page endings. In order to do this, both the Mac and

the PC must use the same fonts. And at the same time, the computer industry generally is now moving to scalable outline font technology.

Even though Adobe will publish the specification of the PostScript language, the information as to how Adobe itself implemented PostScript, such as its methods of halftone screening, will not be published. Adobe spent a lot of time and money on developing its implementation of PostScript, and it will continue to protect its trademarks strenuously. Even though Adobe has invested a lot of money in the technology, preservation of the standard by publishing the specification of the language fonts was more important than guarding the previous secrecy. Adobe will continue to protect its rasterizing technology, however.

Both Apple and Microsoft are part of the joint development for new printer software in addition to the new font effort. Microsoft acquired PostScript compatible printer software when it bought a company called Bauer Enterprises in July 1989. Bill Gates, Microsoft, admitted that the Microsoft-Apple version of PostScript may well develop in a different direction from Adobe PostScript. He said that they will plan to extend the language to improve the printing of bitmaps, for example. Microsoft will look at developments added to the language by Adobe, and while there would not necessarily be two different PostScript standards in the future, it was a real possibility. However, this is the first time that the two companies, Apple and Microsoft, had co-operated on the development of system software.

John Warnock, Adobe's Chairman, said that Adobe was unlikely to build support into its version of PostScript for Royal fonts, the Microsoft and Apple font. He said it was not fair to burden every PostScript printer with the extra cost, when many users may not need it.

According to Warnock, the printing and publishing industry have gone through a very dynamic change over the past five years. Adobe has introduced a very successful de facto standard, and for the first time in the history of printing and publishing, every major typesetter vendor supports the same type format, with a large number of PostScript proofing devices. There are over 60 devices shipping - color, black and white, large format, high resolution, low resolution, film recorders - and another 63 in the pipeline. Warnock believes that this standard is so important to the printing and publishing industry, that he is not going to let it fail in teaming with Microsoft. Warnock said that when the full PostScript specification is published, there will be no excuse for any clone printer to be incompatible. Warnock stresses the need for a single imaging model if the ability to ship documents between different machines around the world in electronic form is to be realized.

In addition to the information about Adobe's font format, Adobe made a commitment to put on public view all of the extension work which it had been working on for the past several years, as it had developed, enhanced, and moved PostScript forward. This included work in the area of truly device-independent color, color enhancement and calibration, and image compression. It also included standardization for the PostScript family through a much higher level of specification for option-handling such as paper sizes, communications, and other practical features.

Chuck Geschke, President of Adobe systems, said that the debate was not just about fonts, but about the status of PostScript as a standard. There was a lot more to the PostScript standard than the particular typeface strategy Adobe had picked. The central theme of PostScript as a standard was the whole notion of document interchange, but what was needed was commonality between different interpreters. Geschke said that the issue was not interchanging text documents, but instead the ability to exchange wholly-integrated visual communication media, that is text, line art, and photographic material, all in a consistent text format.

Jonathan Seybold feels that there will not be two competing forces driving PostScript and two competing font formats, the Adobe standard and the Apple-Microsoft standard. Seybold feels that the competition would be healthy for the industry and for users. Two was probably the optimal number of standards, providing healthy competition without unmanageable chaos. Seybold feels that Microsoft is moving into a driving position in imaging technology, and they needed the support of Apple and Adobe.

To add to the PostScript shakeup, Glenn Reid, Author of Adobe Systems, PostScript Language Program Design, has left Adobe to join Steve Jobs' NeXT, Inc. Reid worked for Adobe Systems for four and a half years, starting as a member of the technical staff and ending as the Manager of Developer Tools and Strategies. NeXT is currently the most significant practical supporter and implementor of Adobe's Display PostScript [ST90c].

For the future, Adobe is planning to make enhancements to PostScript in areas such as halftones, color, image compression, putting intelligence into printers for handling documents as opposed to pages, and for device handling. John Warnock, Adobe's Chairman, said that he would like to make Display Postscript a standard for computer graphics on monitors, the way PostScript nearly is for the printed page [STR88]. DEC, IBM, and NeXT are the primary developers, and Apple has refused to participate. Display PostScript is part of Warnock's effort to create a uniform visual interface for all brands of computers.

Ken Strauss, in a report of a conversation with Warnock in the *PostScript Language Journal*, states that Warnock's idealism must be balanced with practicality. Adobe's attorneys have advised Warnock to stay away from the ISO committee meetings, even though Warnock enthusiastically accepted. Strauss feels that Quality Assurance could present an impossible problem, so long as Warnock is part of Adobe. Although Warnock wants visual interface standards for printer and monitors to be accepted, he does not want to become the enforcer. Enforcing standards could easily become too expensive for Adobe. Enforcement would devour the time of its most creative people and generate law suits [STR88].

In addition, support has become a sore subject with Adobe and its clients. Occasionally, Adobe hears from other developers who are looking for proprietary information or permission to use one of their copyrighted formats. The requests continue to wait, indicating that permissions have been a casualty of the fast growth at Adobe [STR88].

On-demand publishing is becoming a popular term, but it may not be practical in the PostScript language yet [STR88]. Problems with reliability of volume printers using PostScript is caused by the highly complex interaction of the programming, special sensors, and delicate machinery. Jam control and error control are very difficult, and handling them efficiently is a particular problem with duplex printers and printers that process more than one page at a time.

To recover from any error, the PostScript controller must know exactly where each piece of paper is at all times. The controller must also detect the point in the printing process at which the error occurs. Any sheets that have been downloaded by the computer since the sheet containing the error was printed must be discarded. The job must be restarted at the page containing the error, without involving the user.

Restarting the job automatically means that a great deal of information must be maintained in the memory of the printer. Either the PostScript commands that make up each sheet or the bit image of the sheet must be stored until the piece of paper has progressed through the printer. If there are many sheets moving through the printer simultaneously, the sheets take up a great deal of memory. Still more memory is required to track the position of the sheet belonging to each image at any given instant.

Working with PostScript can be frustrating because the tools are still limited. New users even have difficulty finding and accessing some of them, for example, the downloadable error-handler, which operates in the printer.

Adobe's error-handler is not part of the PostScript ROMs for purely historical reasons (that was the way development unfolded). Now, its inclusion may depend partly on the price of the ROM.

5.4.2 EPS

Encapsulated PostScript (EPS) format is an interchange standard for file format, that is, the structure of the data used to record an image onto a disk. EPS files are combinations of the PostScript code that tells the printer how to print the image and a PICT image that tells the screen how to display it. EPS can be based on ASCII or binary subtypes, and is a displayable PICT image used for storing object-oriented artwork.

PICT (from *pict*ure, not an acronym) is the oldest generic file format on the Macintosh and is used to transfer the object-oriented graphics created by draw programs. It can hold any mix of bit maps and resolution-independent objects, which are encoded in QuickDraw, the Macintosh's native graphics language. The original PICT objects and bit maps only allowed eight colors: white, black, cyan, magenta, yellow, red, green, and blue; however, the current standard, PICT2, allows for full 32-bit color, which is the most the Macintosh can produce [AKE87]. The PICT format can hold bit maps with resolutions greater than 72 dpi, but some applications may convert high-resolution bit maps back to 72 dpi [PAR90].

PICT is suitable for medium-quality line art and low-resolution bit maps (which can reside in the same file) with a very limited color range. Unfortunately, there is not way to tell if a PICT file contains bit maps, objects, or both until you open it. When graphics are cut or copied to a clipboard for pasting into another application, they are normally converted to PICT format [PAR90].

PICT2 is an extension of the PICT format, and it has two subtypes: a 16.8-million-color version, commonly called 24-bit PICT2, and the more prevalent 8-bit PICT2, which holds only 256 colors. As with PICT, the PICT2 format sets no limit on the resolution of bit maps except that which is imposed by the creating application [PAR90].

With 8-bit PICT2, a custom 256-color palette can be saved along with the image data, enabling any PICT2-wise application to recreate the original screen appearance. Unfortunately, come programs don't save the custom palette when exporting an image, and some importing applications ignore the custom palette. In these cases, when the image is opened by the new application, its colors will be determined by the current color palette, which can wreak color-shift havoc. Color shifts may also occur when a chunk of 8-

bit PICT2 from one document is pasted into another 8-bit PICT2 document [PAR90].

PICT2 is usually a better choice for work involving presentations, in which the final image is viewed on-screen or on a slide than it is for desktop publishing. In general, PICT2 is readily imported by publishing applications. Many page-layout programs offer contrast and brightness adjustments for gray-scale art but not if it is PICT2 format. PICT2 is also largely shunned by color-separation applications [PAR90].

Bit-mapped images can be stored in MacPaint, PICT, TIFF, or EPS formats. EPS files are usually very large files and their size needs to be considered in the mechanism of tool interfaces for the systems engineering environment.

5.4.3 IDL

The technology for the Interface Description Language (IDL) has its roots in compiler technology and provides support for tool intercommunication. IDL technology emphasizes generation of batch-oriented applications rather than supporting interactive applications. The IDL technology effort grew out of the PQCC (Production Quality Compiler Compiler) project at Carnegie-Mellon University, and was developed as a second generation to the compiler research support system LG. In the context of the Catalyst, IDL technology would be classified as a tool interface or a tool to tool interface.

The IDL technology consists of two parts:

- Language
- Generation tool

The first component of the IDL technology is the language itself, and is called IDL. The IDL was developed as a generalized data definition language to permit interfacing of results from different project components. IDL is an intermediate language that represents source code. IDL was used to define the intermediate representation for Ada™ called DIANA (Descriptive Intermediate Attributed Notation for Ada). [3] DIANA, developed by Intermetrics, Inc., was designed to be especially suitable for communication between two essential tools, that is, the Front and Back Ends of a compiler as

[3] John Faure (SPS) states that Intermetrics implementation using IDL may not be efficient with respect to memory utilization. Even though complexities of IDL exist, an efficient implementation is possible, however.

well as useful by other tools in the Ada support environment. [4] IDL was also used as the core technology for the compiler automation tool set developed by Tartan Laboratories.

A full and detailed description of IDL is found in [NES86]. IDL uses a formal notation for structural and constraint descriptions. From these descriptions both generation tools automatically produce software for reading, writing, and manipulating instances of the described structures, as well as for checking specified constraints on information contained in the structures. One of the interesting properties of IDL is that the semantic specification does not constrain the implementation strategies which may be used to achieve it. With the combination of the small design, the formal model, and the flexibility of choices, gives IDL some promising directions for research based on the IDL model.

The second component of IDL technology is a generation tool. Lamb discusses the generation of software based on a IDL description and the tool supporting the generation in his Ph.D. thesis [LAM83].

IDL was developed as an alternative approach for supporting the development of attributed graphs for prototypes and to overcome drawbacks of traditional database approaches. For example, the traditional database approach is a viable option for large, complex system development but with an implementation in Ada it has its problems:

- The classes of graphs that are to be manipulated are described using a database schema, which although not similar to Ada, is relatively easy to use.
- Operations are provided by dynamic redefinition in experimental development, but for large development efforts this reduces managerial control as well as sharing among development groups because data objects are not well documented.
- Lack of support for user-defined types and the run-time overhead of such systems.
- Data base systems usually only provide support for a limited set of predefined types such as integers, reals, and character strings. In a software development environment, much richer types are needed.

[4] The Front end process(es) are defined as all procedural steps (e.g., scanning, parsing, etc). CASE products for the non-code producing steps are called front-end tools. The Back end process(es) are defined as the code generation and optimization steps in a compiler or 4GT environment. By definition, the second phase (back-end) implements the specifications.

- Run-time overhead for the newer, more powerful data base models, such as the relational approach, is too prohibitive.

IDL has become a popular alternative approach to addressing the limitation of current database systems. IDL is a BNF-like language for describing attributed graphs. The IDL processor takes an IDL description as input and creates a set of Pascal procedures that realize some of the operations available in interface packages generated by a software tool called GRAPHITE. The IDL processor has been rewritten by several organizations so that it produces an Ada interface package. None of these organizations has designed the interface package so that it is supportive of experimental development. IDL is more ambitious than GRAPHITE and is intended to be language independent and provides a number of additional capabilities. Table 5.4.3-1 summarizes an overview of the features of IDL.

Table 5.4.3-1. IDL Language Features

Features	IDL
Application Architecture	structure manipulation and read/write primitives
Primitive Types	boolean, integer, string, rational
Composition	set, sequence, class
Constraints	graph oriented assertion language
Generated Code	direct encoding, table-driven
Generation Tools/ Environments	collection of off line-oriented processing tools

The following are some important issues here for language designers, compiler implementors, and environment implementors:

1) When programs are generating the record or structure definitions, bit-by-bit layout control may be required. Sophisticated compilers which give the user no layout control and already do optimized packing, make interfacing extremely difficult.

2) Mechanisms which are intended to prevent errors at the interface level must have ways of being turned off by sophisticated tools which have the precise knowledge of what constitutes a change.

3) To date, realizations of the IDL model (to utilize a system of a variety of languages on a variety of machines to communicate with each other, while still maintaining efficiency) have dealt with short-lived data. The data was encoded in the programs and not in a database environment. The data description must eventually live with the data, and the IDL symbol table must be part of the information transmitted. This appears to be a interesting and promising area of research.

4) The area of attribute grammars based on IDL to create object-based database systems is an area for new research.

5) The use of graphical input specification for complex data structures is an area that needs to be explored.

The possible future direction of the technology for IDL-based tooling and the IDL language itself in reference to research, development and/or engineering ideas needs to address the following:

- Design of target languages supporting the IDL type model
- Provide support for separate compilation of IDL specifications
- Provide support for resolution of issues of use of IDL in large system construction
- Design automated design support tools for the construction of complex IDL definitions
- Provide graphical data presentation for specifications and debugging
- Design support for persistent data objects
- Use and adapt techniques to make robust data design with good performance

As on-going research, in 1985, the SoftLab project at the University of North Carolina has been building a UNIX™/C-based IDL generation tool. This translator uses a table-driven implementation for the generated code. Two debugging tools are provided with the SoftLab implementation. On-going research needs to continue to make IDL a viable solution to interfacing system to tools and tools to tools.

IDL may be useful for converting between complex, structured data representations, but not recommended for converting between simple

formats like those used by a word processor (i.e., Wordstar to Microsoft format, SQL to Object-Base format, EDIF to SFS Standard format, etc.).

5.5 Document Representation with SGML

The Standard Generalized Markup Language (SGML), ISO 8879, defines a highly portable system-independent standard for exporting and importing documents. SGML was published as an international standard in October, 1986 [DUR90]. SGML specifies a method for describing the content, structure, and organization of an electronic document, and was designed for editing, publishing and printing. It does not provide support for describing the formatting or rendering of the document.

SGML describes a document by providing a mechanism for tagging parts of a document according its function within the document. For example, SGML can be used to indicate that a part is a title or a section heading, but it does not specify whether it is to be set in boldface or located at a particular location within a page. Standard forms based on SGML are context-free grammars that describe documents as hierarchical objects. SGML provides for document components, such as footnotes, which do not fit strictly within a hierarchical model. SGML documents may incorporate almost any kind of data, including scanned images and executable code.

SGML's rigorously defined structure and explicit tagging of structural elements make conversion of documents to and from SGML relatively easy. Proponents of SGML believe it will have wide application for open systems and multimedia documents. SGML is already finding use in such diverse areas as music and hypertext.

SGML has been promoted by the Association of American Publishers (AAP), and its support is broad-based. The AAP has developed its own SGML-based syntax for representing electronic documents, including textual, tabular, and mathematical components. The international standards organizations have approved SGML as a standard, and the National Information Standards Organization has adopted it as a standard. The U.S. Department of Defense, Internal Revenue Service, Department of Health and Human Services, and National Institute of Science and Technology have all endorsed SGML as a standard [RUD89]. The Computer-Aided Acquisition and Logistics Support (CALS) Steering Group is continuing the development of a CALS standards framework to include SGML. U.S. committees are attempting to reach consensus on SGML and other standards in the framework before submitting them to the International Standards Organization [SMI90]. Document processing is provided by SGML through the specification of FIPS 152 for the NIST-Application Portability Profile (APP).

6 Virtual Machine Interface Standards

6.1 POSIX

The standard Portable Operating System Interface for Computer Environment (POSIX) is defined as a set of IEEE standards (1003.1) for an open system environment to allow portability of application programs between operating systems. These set of standards specify a standard operating system interface and environment to support application portability at the source code level. It is designed to be used by both application developers and system implementors.

POSIX is not, as some may think, an operating system, but an interface between applications programs and an operating system environment. POSIX specifies a set of external calls and interfaces for operating systems. If an application makes POSIX calls, it can be recompiled on any POSIX-compatible operating system and will run unchanged. The POSIX calls are restricted to being written in a portable language without making any non-POSIX operating system calls. POSIX does not apply only to UNIX; DEC has promised to release a POSIX-compliant version of VMS. Therefore, POSIX provides a mechanism for applications to run on heterogeneous systems.

The term POSIX refers to both the initial P1003.1 document and to the entire P1003.x family of efforts which are intended to promote the portability of software applications from one machine to another. There are several approved POSIX working groups with several more planned or anticipated. Their efforts include P1003.1 as the initial POSIX standard; P1003.2 is addressing shell and application utility interfaces; P1003.3 is defining the test method specifications for testing conformance to the P1003.1 standard; and P1003.4 is addressing real-time extensions. The POSIX committees are defining a set of standard UNIX interfaces with both "C" (P1003.1) and Ada bindings (P1003.5), standard shell and tools (P1003.2), and various UNIX extensions like real-time and security.

The original work on a UNIX standard was begun in 1981 by /usr/group which completed the initial work in 1984. In January 1985, the P1003.1 working group was formed under the leadership of James Isaak to develop a standard for portable operating system environments. The /usr/group document was used as the starting point for that effort. POSIX was approved an an IEEE Trial Use standard in March 1986, and was adopted by the IEEE and NIST as Federal Information Processing Standard FIPS-151 in 1988 [MAR87].

NIST brought the standard into line with the latest IEEE version, in May 1989, with the release of FIPS-151-1.

In March 1990, POSIX was mandated to be used for operating systems that are either developed or acquired for Government use where POSIX-like interfaces are required. This FIPS standard is applicable to the entire range of computer hardware, including micro-computer systems, mini-computer systems, engineering workstations, and mainframes [USA90].

The goal of the IEEE POSIX committee is to define a portable, open systems environment. This committee has the backing of the major standards-making organizations, vendors, and users. The Open Software Foundation, which includes sponsors from all major computer vendors, has established the specification of a UNIX standard to promote increased software application portability as its goal.

The POSIX standard is comprised of four major components [IEEE88]:

- Terminology, concepts, and definitions and specifications that govern structures, headers, environment variables, and related requirements
- Definitions for system service interfaces and subroutines
- Language-specific system services for the C programming language
- Interface issues, including portability, error handling, and error recovery

Initially, the focus of the standard is to provide standardized services via a C language interface. Future revisions are expected to contain bindings for other programming languages as well as for the C language. This will be accomplished by breaking the standard into two parts, a section defining core requirements independent of any programming language, and a section composed of programming language bindings [IEEE88].

The core requirements section will define a set of required services common to any programming language that can be reasonably expected to form a language binding to this standard. Theses services will be described in terms of functional requirements and will not define programming language-dependent interfaces [IEEE88].

Language bindings will consist of two major parts. One will contain the programming language's standardized interface for accessing the core services defined in the programming language-independent core requirements section of the standard. The other will contain a standardized interface for language-specific services. Any implementation claiming conformance to

IEEE Std 1003.1-1988 with any language binding shall comply with both sections of the language binding [IEEE88].

The P1003.1 POSIX C language binding was published as IEEE Standard P1003.1-1988, was accepted as FIPS standard 151-1, was recognized as an American National Standard by ANSI, and was recently adopted as ISO standard 9945-1. All POSIX standards , including this Ada binding, are targeted for those same standards bodies.

The IEEE P1003.5 POSIX Ada Binding is going to ballot in August 1990. The P1003.5 working group is seeking as large a balloting group as possible for this draft IEEE and ISO standard. Draft P1003.5 represents the first Ada binding to POSIX, since the original interface binding was for the C language. In developing the binding, the working group has abstracied the intention of the original binding into well structured Ada with good object-oriented definitions, while still allowing efficient implementations. As the first Ada binding, this standard will provide the basis for future Ada binding efforts to the other proposed POSIX standards, and this effort will have significant influence on bindings for other languages and the development of language-independent standards. The P1003.5 Working Group plans to resolve all comments and objections for parties of interest and the members of the balloting group.

POSIX, as currently defined, is the crucial first step in providing a vendor independent interface specification between an application program and an operating system. The current definition, however, must be extended in order to provide interface specifications for full operating system functionality. The following areas are outside the scope of this standard [IEEE88]:

- User interface (shell) and associated commands
- Networking protocols and system call interfaces to those protocols
- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability
- The behavior of system services on system supporting concurrency within a single process

Consequently, these additional extended interface specifications must include [USA90]:

- Shell and Tools: These functions provide an interactive interface for users to control processing, e.g., listing files in a directory.
- Advanced Utilities: These utilities provide additional capabilities and specialized functions that make users and programmers more productive, e.g. full-screen editing.
- System Administration: These functions are required to operate and maintain the system, e.g., mount a file system.
- Terminal Interface Extension: These functions are called by application programs. They enable programs to perform interactive terminal operations in a way that is independent of the type of terminal being used, e.g., turn on attributes such as blinking characters or reverse video.

POSIX, when fully extended, will provide the functionality required to support source code portability for a wide range of applications across many different machines and operating systems. However, even the extended POSIX will not be sufficient to achieve portability for all applications [USA90].

POSIX will provide the basis of an evolving an architecture for applications portability. As additional standards are developed which specify the interface bindings for functional areas such as graphics, data bases, communications, file systems, user interfaces, and other languages, POSIX can be seen as the hinge for the resolution of the issue of heterogeneity of machines.

There is increasing recognition of the need for an architectural approach to applications portability. This recognition has come about because earlier attempts to use a language-based approach to applications portability were not successful. Language portability is only one aspect of the problem of porting applications software from one operating system environment to another. Applications software portability depends on additional factors which include [USA90]:

- Characteristics of the underlying hardware/software (e.g., word length, input/output (I/O) architecture, processor, operating system)
- Portability of software utilities used by the application (e.g., database management, graphics, operating system functions, and communications)
- Data form, format and representation that may need to be transported with the software

- Language implementation (compiler/interpreter/processor) including specific limits or subset of the language used in programming (e.g., magnitude of numeric values, number of subscripts and number of labels)

Unless each of these factors is addressed as part of an overall architecture, the benefits of applications portability will not be fully realized [USA90]. A planned Applications Portability Profile (APP), FIPS 151, has been developed to provide sufficient functionality to accommodate a broad range of application requirements.

Table 6.1-1. Applications Portability Profile [USA90]

Function	Element	Specification
Operating System	Extended POSIX	FIPS 151 1 (IEEE Std 1003.1-1988) IEEE P1003.2 (proposed FIPS)
Database Management	SQL IRDS	FIPS 127-1 FIPS 156
Data Interchange - Graphics - Product Data - Document Processing	 CGM IGES & PDES SMGL ODA/ODIF	 FIPS 128 NBSIR 88-3813 FIPS 152 ISO/IS 8613
Network Services - Data Communication - File Management	 OSI TFA	 FIPS 146 (GOSIP) IEEE P1003.8
User Interface	X Window System	Version 11, Release 3 (proposed FIPS)
Programming Services	C COBOL FORTRAN Pascal Ada	X3J11/88-002 FIPS 021-3 FIPS 069-1 FIPS 119 FIPS 109

The functional components of the APP constitute a "tool box" of standard elements that can be used to develop and maintain portable applications as shown in Table 6.1-1. A key aspect of the APP is that it is an open systems architecture based upon non-proprietary standards. The current planned components of the APP are summarized in and described in the following paragraphs [USA90].

Database management is an important aspect of applications portability. A growing number of organizations use a Database Management System (DBMS) to allow application programs, written in a variety of languages, to work on the same basic data. In addition, a DBMS can facilitate language independence in the design, development, and maintenance of data resources. FIPS 127-1, Database Language SQL, and FIPS 156, Information Resource Dictionary Systems (IRDS), are the initial components to meet the database management requirement [USA90].

In addition to the mechanism for managing the data, the data itself is an important aspect of applications portability. In many situations, the problems associated with porting the applications software from one system to another pales in comparison to the problem of porting the data. There are three categories of particular concern regarding **data interchange** [USA90]:

- Graphics
- Product Data
- Document Processing

In the area of data interchange, the Computer Graphics Metafile (CGM) FIPS 128 is the initial component to meet the graphics requirements. Efforts are underway to extend CGM to include picture-transfer requirements of such areas as engineering drawing and technical illustration, graphic arts, and technical publishing. CGM is one of the standards defined CALS to deliver two-dimensional engineering drawings. Also to be considered are the graphics requirements for office systems such as ODA systems [CHI88].

Initial Graphics Exchange Specification (IGES) and Product Data Exchange Specification (PDES) are the initial components to meet the requirements to exchange product data. IGES has been identified as a standard for representing vector graphics in CALS. Data interchange of product data is provided by IGES through the specification of NBSIR 88-3813 in the NIST-Application Portability Profile (APP). The Standard Generalized Markup Language (SMGL) FIPS 152 and Office Document Architecture/Office Document Interchange Format (ODA/ODIF) are the initial components to meet the requirements for document processing [USA90].

There are two basic network services that should be provided, **File Management** through the element of Transparent File Access (TFA). File Management is an integral part of most applications. File management functions have traditionally focused on accessing data within a local file system. That focus has now shifted to functions that permit shared access to files in a heterogeneous environment of computer hardware, software, and

networks. A standard approach to managing this shared access to remote files is an important aspect of software portability. Failure to provide shared access to remote files will inevitably lead to local, incompatible approaches that inhibit application portability. Transparent File Access (TFA) is the initial component to meet file management facility requirements (IEEE P1003.8) [USA90].

Data Communications facilities permit interoperability among applications in a heterogeneous environment of computer hardware, software, and networks. The requirement to manage shared access to remote files is just part of a large requirement for applications software to perform its functions in a network environment. Failure to provide this function will inevitable lead to local, incompatible approaches that inhibit applications portability. Government Open Systems Interconnection Profile (GOSIP) is the initial component to satisfy the data communications requirements [USA90].

The most neglected aspect of applications software portability is the requirement to maintain a consistent **User Interface** across all systems on which the application resides. The fact that the application is likely to be distributed over a heterogeneous environment of computer hardware, software, and networks means that the user interface facility must provide the flexibility to allow the user to interact with programs within such an environment. The X Window System is the initial component to meet user interface requirements [USA90].

The most emphasized aspect of applications software portability is the requirement for **Programming Language Services**, that is, portability from one system to another. The major problem is that programming language portability is often equated with applications software portability. A key requirement for programming languages is that a sufficient variety be included to encompass the full range of application requirements. The C language binding is the initial component for programming language interfaces. Additional bindings are being developed for FORTRAN, COBOL, Pascal, and Ada [USA90].

The components of the APP represent varying stages of maturity. Some have not been introduced into the formal standards process (i.e., X Window System), other exist only as draft standards (e.g., POSIX), and other have been adopted as national and international standards (e.g., SQL). As these standards mature, there will be a need to update the APP to reflect the changes that will occur. The NIST will establish a process to ensure that the APP incorporates the evolving (maturing) consensus of the national and international standards activities for each of the functional components of the APP.

A newly formed committee, P1003.0, will be examining POSIX in view of other existing or emerging standards to ensure the seamless integratability of these standards. Some of the standards under consideration include Structured Query Language (SQL), Information Resource Dictionary System (IRDS), Graphical Kernel System (GKS), Computer Graphic Metafile (CGM), Initial Graphics Exchange Specification (IGES), and Office Document Architecture/Office Document Interchange Format (ODA/ODIF).

Committee P1003.0 will also be examining the languages C, COBOL, FORTRAN, Ada, Basic, LISP, Forth, PL/I, and Pascal. Data communications services of interest include OSI-based Government Open Systems Interconnection Profile (GOSIP), SUN Microsystems' Network File System (NFS), and AT&T's Remote File System (RFS). In addition, committee P1003.0 will consider administration utilities, international codes and character sets, publishing standards, and user interfaces. The goal of POSIX committee P1003.0 is to specify the standards to be included in a complete environment.

Committee P1003.0's strategy is to first consider international and national standards. If no such specification exists for a particular requirement, the next preferred choice is a standard under development. In the absence of these two cases, the committee will examine publicly-available specifications that are supported by multiple vendors. Examples of interfaces in this category include MIT's X-Windows, SUN Microsystems' NFS, Apollo Computer's distributed Network Computing System (NCS), and IBM's System Application Architecture (SAA) [RUD89].

The IEEE POSIX working group has failed to reach a resolution as of the summer of 1990, on choosing a standard graphical user interface for UNIX systems. Proponents of AT&T/Sun Microsystems' Open Look and OSF/Motif are still locked in a marketing tug-of-war. The committee's deadlock is typical of more than a year of IEEE activity in seeking a standard graphical user interface, as proponents of Motif and Open Look refuse to compromise on their favored GUI. Mehta, a UniSys marketer and chairman of IEEE P1202, which is standardizing windowing environments, said he hopes to see the committee's mission changed [WAG90]. He would like the goal to be a common, or 'virtual,' API, a standard that developers could write to make it easy to migrate an application to Motif, Open Look, Microsoft Windows, or the Macintosh user interface. Mehta said he hopes to see a new charter, or 'project authorization request,' in IEEE jargon, reflecting that mission change be approved within three to six months, and a virtual API selected within a year thereafter. That API would be based on technology now on the shelves for the Extensible Virtual Toolkit, or XVT, from XVT Software, Boulder,

Colorado [WAG90]. Some say that the virtual API definition is not clear enough for presentation to the Standards Evaluation Committee (SEC), and will continue to be refined for the proposal. The virtual API standard is probably three or four years away.

The DoD is enthused by POSIX because it is not proprietary in any way, so specifying POSIX in a contract is virtually protest-proof. Consequently, POSIX is more appropriate for contract solicitations that UNIX, which was developed, trademarked, and essentially controlled by AT&T. Competitors have argued in the past that AT&T had an unfair competitive advantage whenever solicitations called for UNIX or UNIX-like features. This relieves the pressure for government purchasing agents who write specifications; there is no opportunity to get sued for choosing POSIX, because it's government-blessed already, and supposedly neutral in that it doesn't favor specific vendors.

Others believe that it is not clear the POSIX is emerging as a standard [KEL90]. Donn Terry, chairman of IEEE's committee on the POSIX operating system interface standard, and a project manager at Hewlett-Packard Co., Fort Collins, Colorado, says that the DoD has a mind of its own, and each service has a mind of its own, too. But he admits, it will be hard for the DoD to ignore these differences. The problem is the fact that POSIX, FIPS 151-1, is not a complete operating system specification as an interface. If the user provided just those interfaces in FIPS 151-1, and nothing more, it is useless. It provides no command and no systems administrative types of things. The next move belongs to the IEEE whose committees are working on sub-standards to bind POSIX to high-level languages, such as C, FORTRAN, and Ada. The command language information should be out late 1990 or early 1991, and there will be a FIPS that grabs a preliminary draft of it at the end of 1990. The IEEE committee process takes time, which isn't always good for industry. In Terry's opinion, it creates chaos for the market place because you have two subtly different standards out there for a couple of years; the final IEEE standard, and the initial FIPS [KEL90].

In spite of these differing opinions, the Navy's Next-Generation Computer Resources program (NGCR) will base its standard operating system on POSIX. The move is the latest in a series of Pentagon actions making POSIX the military standard for an operating system interface. The message to industry is "if you're not doing POSIX, you're not doing business with DoD [KEL90]." Roger Martin, manager of the software engineering group at the National Institute of Standards and Technology says its not a matter of is this going to happen, rather, it's a question of how long will it take to get things done [KEL90].

The Navy's NGCR action followed the release of "Air Force Communications-Computer Systems: 1990 Planning & Architecture Guidance," an Air Force directive, that all but mandated POSIX. To achieve objectives of hardware independency and software portability in the Air Force, the Air Force is transitioning to POSIX as stated in their documents. All new operating system environment acquisitions should specify POSIX [KEL90].

POSIX doesn't meet all the Navy's needs for mission-critical systems; however, Navy officials use the standard as ' a roadmap' for NGCR according to Cmdr. Rick Barbour, program manager for the NGCR operating system effort [KEL90]. The project team will find out where POSIX requirements are, where they meet our requirements and where they don't, and between now and 1995, they will work with POSIX and will start their mil-standard effort. The Navy would like to see a move to make POSIX mandatory as are all the services. The Army is moving toward POSIX also as indicated by the service's Sustaining Base Information System Project which specifies POSIX as its operating system interface.

While the National Computer Systems Laboratory (NCSL) is pleased with the progress of the POSIX FIPS efforts, there remains much to be done to make the POSIX a robust standard. Extensions to the POSIX standard must be introduced to attain added functionally. NCSL is proposing or evaluating extensions to the POSIX standard shell and tools, system administration, and terminal interface. Additional functional areas being worked on within the Applications Portability Profile (APP) include user interface and network services [HAL89].

Each of the APP areas will follow the standardization process used for the FIPS 151. A proposed FIPS will be brought forth, based on a stable draft that likely will not be the final draft used for the IEEE standard. The goal is to, again, give the agencies a FIPS to use, and vendors a reasonable target to shoot for. The intent is to bring these initial FIPS in line with the national and international standards when they are completed, and to agree on one uniform set of standards throughout the industry. In 1987, the POSIX standard was submitted to ANSI for approval and to ISO for registration as a Draft International Standard.

To accomplish this, NCSL will continue to hold workshops for implementors, vendors, and users alike; to make proposals; garner feedback from participants; and determine the desired direction to take regarding these extensions and other concerns. A related effort to the standards development, and one critical to its success, is the parallel development of a

test suite to test conformance to the standard. Unless it can be shown that a given product conforms to a standard, that standard does little good in a procurement analysis [HAL89]. The National Bureau of Standards is developing a reference implementation test suite which will test conformance of an operating system environment to the POSIX FIPS. This test suite is being developed based on the test assertions and specifications contained in the P1003.3 draft document.

6.2 UNIX™

The UNIX Operating System is a working environment of considerable power and effectiveness. It is considered a first generation environment, having a low level of definitional support and enforcement [RUD89]. It was originally designed by software engineers for the specific task of software engineering. Today there are over half a million programmers using some form of UNIX on over a hundred different computers for diverse applications in the business, industrial, educational, and scientific communities [CUR87].

The single most noteworthy factor in the evolving UNIX story is that it has gradually come to be regarded as the 'lingua franca' or common foundation for development work across several different computer architectures. For example, the concepts of standardization and commonality apply to a fundamental design rule of UNIX know as 'device independence.' This means that UNIX users can choose the peripheral best suited to their particular application and not be constrained by the nature of the device database or the architecture of the operating system. This is, of course, only one example of flexibility of UNIX as a development platform, and illustrates the power inherent in choosing an open systems approach to development work.

Because of its popularity as an open and extensible system, UNIX is available on more different types of computers than any other operating system. The migration to UNIX is accelerating. UNIX is now available on a broad range of engineering workstation, superminis, desktop micros, and mainframe computers.

Sun Microsystems has based its workstation philosophy on standards and open systems architecture. The Sun workstation philosophy is compatible with the UNIX paradigm of open, extensible systems, especially relevant when we consider the trend towards 'standardization' of UNIX. William Joy, a founder of Sun Microsystems, is generally regarded as 'The Father of Berkeley UNIX.' While attending Berkeley as a graduate student, Bill Joy played an important role in the addition of substantial new functionality to the UNIX system. Today, Sun and AT&T are working together to provide a

converged version of UNIX which will incorporate the best features of the Berkeley and AT&T versions.

The trend toward ultimate standardization of UNIX will, itself accelerate in the next few years. This acceleration will be spurred by renewed interest in UNIX as a unifying force within the commercial and data processing community, and by technology advances in the areas of personal computers and workstations for technical software engineering tasks.

There are four primary technology areas being developed in the UNIX realm which will exercise major influence on the state-of-the-art of software engineering.

- Multi-Processing
- Window System Technology
- Database Technology
- Artificial Intelligence

Advances in **multi-processing** technology are providing software engineers with faster, more powerful computing engines, often at significantly lower cost than previously available computers. The trend is for increased 'horsepower' at a reasonable price. Faster processing speeds and performance optimization allows software engineers to work on applications which were previously 'speed limited' or too 'speed dependent' to be feasible.

Recently, more concrete technology advances have been made in the **window system** area than in any other. Adaptable, extensible, easy-top-use, bit-mapped window systems have been high on the software engineers 'wish list' for many years. After an initial, bewildering variety of choices, window system technology has matured and is undergoing its own 'standardization' process.

The potential for **database technology** to solve some of today's most complex CASE problems remains largely unfulfilled. It is becoming generally recognized that relational databases provide only limited serviceability or applicability for complex CASE environments. A powerful database for CASE applications must support management of all of the objects in the development environment. With no coherent, centralized management of these objects, there will be anarchy. Furthermore, CASE databases must support the provision of different 'views' of objects to accommodate the diverse needs and perspectives of the different roles in the CASE environment. Because of these complex issues, the solution will be a difficult one.

A new paradigm is required for CASE specific databases. Rather than regarding a database solely as a convenient manager of all objects in the development environment, it would be better to focus on the fundamental issue in CASE, integration [CUR87]. The new paradigm must include not just the management of all the objects, but the method for effective communication of all the information in a large software project. The complexity of output from differing tools, to iterations and cycles within individual phases of the development process, the difficulties of heterogeneous, networked environments, all combine to exacerbate the problem. Developers, managers and programmers are spending most of their time gathering the bits and pieces of data relevant to their particular view of the development activity.

Object-oriented and associative databases hold the solution. Communications in a large environment is critical, so a relationship manager may need to be constructed. The mechanism of a networked relationship manager would not only link the diverse outputs from the tools employed during the project, but would facilitate the notion of different views [CUR87]. Object-oriented and associative database technology is receiving a great deal of attention in the software engineering community, and it is clear the database technology is both a pacing and a leading edge technology in the integration of large scale software environments.

Artificial intelligence technologies hold one of the most exciting prospects for the future of software engineering. With the arrival of powerful, cost effective workstation, AI technology has become more affordable on a cost-per-seat basis. The marriage of AI and CASE, called I/CASE at Sun Microsystems, may help to solve software engineering problem encountered throughout the software life cycle. For example, AI based applications are generating code, building and implementing test routines, assisting and optimizing database performance, executing specification, tracking project status, and providing 'help' to assistance to programmers. I/CASE will lead to tools and techniques that allow development work to at last keep pace with the developers' 'stream-of-consciousness.' This technology will also allow for more sophisticated testing and checking during the development process. In the future, we will also see improvements in reusability, automatic program synthesis, and intelligent databases resulting from this technology.

With these technologies integrated into the UNIX programming environment more and more of the software engineering tasks can be shifted to the machine. A wider spectrum of tasks that the user can expect the machine to do becomes available. Eventually, the user will arrive at a fully automated life cycle where all the tasks and objects in the software

development process are coordinated by the computer itself. Even now, a subtle level of integration is occurring through the interaction of window, code browsers, databases, and programming tools. This represents integration and automation of the 'programming-in-the-small' activities where UNIX has always excelled. The next goal is to extend these technologies to 'programming-in-the-large' throughout the software life cycle.

UNIX is a accepted as a functionally mature technology. There are not standard versions of UNIX, but POSIX is capable of converging the UNIX version variants. Some areas still exist for UNIX to address, for example, fault tolerance, better error recovery on the network level, more business-oriented applications rather than scientific and engineering applications, and graphics applications.

UNIX is gaining increasing recognition as a favorable operating system for portability and interoperability, and fits will into NCSL's proposed Applications Portability Profile (APP) [HAL89]. For example, in August 1990, the Army purchased multiple 386 Prime Computer PCs to run UNIX. The systems are required to support up to 16 terminals each, and be compliant with POSIX and GOSIP. If the units pass the Army's acceptance test at the Information Systems Engineering Command at Ft. Huachuca, Arizona, the Air Force and Navy will be buying the same equipment also. The contract has a potential value of $700 million [HAR90].

UNIX is chosen as an underlying virtual machine for several reasons [HOU87]:

- Primitives
- Tools
- Interface
- File System

UNIX **primitives** are few in number and easy to define, and therefore, UNIX is available on many machines. Portability is not a major obstacle since most of the UNIX tools are written in the C language and C is available on every UNIX platform.

UNIX contains over 100 **tools**. These tools can be used as building blocks to the overlying system environment, thus eliminating much of the development of basic underlying system environment functions. Each tool is invoked with a different command. Most of these commands have a series of parameters that are associated with them. The combined functionality of

these tools makes UNIX a very powerful environment, but unfortunately most users of UNIX only tap a small part of this power.

The UNIX **Interface** is called the shell. The shell is in many respects a very high level language (VHLL), because it allows the user to use tools within the environment as objects. This is accomplished in a manner similar to a programmer's use of variable in a high level language. The output from one tool can be directed, or piped, as input to another tool. It is this capability that makes the underlying UNIX tools suitable for building blocks to an overlying system environment. Raymond Houghton, in *ACM Software Engineering Notes*, argues that UNIX is more suited for building that it is for direct use due to the resulting unfriendliness of the interface [HOU87].

The UNIX File System can be used as an underlying database because the UNIX files are defined as strings of characters. The shell allows the file system to be defined hierarchically, and the files can be easily manipulated. Consequently, the products of software engineering can be easily tagged, stored, and retrieved from the file system in a manner similar to a hierarchical database.

7. Updates to Interface Standards Studies

7.1. GOSIP

Now in 1992, several key federal agencies, including the Department of Defense, NASA, and the U.S. Department of Agriculture, are taking a harder policy line on GOSIP technology, with RFPs either recently released or on the way. Previously mentioned as an afterthought, GOSIP is now required as the foundation for new computer and networks in these recent documents. GOSIP version 2 becomes mandatory in October 1992, adding terminal emulation to OSI and integrated services networks. Version 3 will be released summer of 1992 as part of the Industry/Government Systems Specification (IGOSS).

IGOSS is an industry- and government-backed procurement that combines manufacturing protocols and GOSIP in a unified OSI model. GOSIP Version 3 becomes mandatory in 1994 along with X.400 and X.500, electronic data interchange standards and Directory Services protocol. The latter is supported by NIST, General Services Administration, and National Science Foundation.

In the two years since its mandate, GOSIP's future has been in question, since some have said the standard has failed to change the face of federal computer systems and networks. However, Sanford George, senior network engineer

and contractor for NASA's Jet Propulsion Laboratory in Pasadena, CA says, "GOSIP is going to happen." He added, however, it is premature to build an OSI-only network, most federal users will support various protocols in the next ten years, including OSI. E.W. Huber, a senior consultant with Hughes Aircraft, Los Angeles, CA, says, "We are still waiting for GOSIP" [HIG92].

A GOSIP movement is afoot; however, NASA has outlined their five-year plan to go to OSI and GOSIP standards along with support for TCP/IP. DoD is calling for OSI protocols in its new Center for Information Management (CIM) Technical Reference Model. The Army has awarded one of the first federal contract that uses OSI as an integral communications tool. The effort primed by Boeing Computer Services, Seattle, WA, will automate the process of tracking, supporting and calling up troops in the Army Reserves and National Guard. Market researchers estimated that the government spent $650 million of OSI-and POSIX-based products in 1991, and that should increase to $1.35 billion by 1996 [HIG92].

Despite these promising GOSIP efforts, today's reality is that most GOSIP endeavors remain on paper or in the laboratory. Products that deals with OSI are scarce, expensive, and often inadequate. IBM and DEC have backed off their OSI stances, citing slow sales. Much of the commercial user sector has sat back and let the federal agencies take the lead in OSI.

7.2. OMG and the Object Request Broker

The Object Management Group had phenomenal growth in 1991 and 1992, by organizing itself along the more flexible lines of an industry consortium rather than a formal standards group [GUT91]. In the last half of 1991, almost every major vendor and large-scale user of object technology began actively supporting OMG. OMG now has well over 100 members and is adding new ones every month. OMG is well ahead of its most optimistic ambitions. Redoubling their efforts on this success, OMG feels that it is the only real chance to short-circuit possible years of unproductive haggling and finger-pointing in the object standards business. In this window of opportunity, its members are committed to effectively deal with the birth pangs of a very important new technology.

This informal and unusual style of organization has given OMG the unprecedented opportunity to act quickly and decisively with a minimum of bureaucratic wrangling and overhead. Small working groups reach many decisions quite rapidly without the exhaustive review processes that would be demanded of a formal standards body. Wisely, OMB has collected interest and support from the existing standards organizations such as OSF, OSI, Unix,

thereby uniting, rather than fracturing, the standards industry around their efforts to promote a common vision of object technology.

OMG endorsed a common object model, an object request broker to support distributed object systems, object databases, and object class libraries for windowing, graphics, memory management, communications, security, and transaction management. A set of object services such as persistence, security, transaction processing, and event handling are planned for endorsement for interoperability among object applications. The architecture of the OMB which defines the core message-handling facility within a distributed object-oriented environment was available as of January 1992, at a prince of $50 [KHE91].

Issues of conformance and interoperability will be resolved by the end of 1992, but the standard was published, nonetheless, so as not to lose momentum. Another issue is the use of compile time/static binding methods vs. run time/dynamic binding method. The compromise reached by the teams involved the adoption of a single higher-level definition language, that will be used to describe the way objects can communicate. The definition language functions in the status method, but a special dynamic invocation interface can to used to specify dynamic binding.

Each member of the OMG actively supports and conforms to OMG endorsements in their own related commercial offerings. Key to the success of the adoption of the OMG standards, OMG's support of handpicked technologies to advance object-oriented systems is projected to lead to quick adoption by users and relatively painless creation of de facto industry standards.

The standard for the Object Request Broker (ORB) was resolved between two camps of vendors, HP/NCR and DEC,/HyperDesk, and Object Design. Final approval by the OMG's technical committee is expected in 1992. Richard Soley, Vice President and Technical Director of OMG, says, "Its an unprecedented level of cooperation and consensus among fierce competitors – the first step to a level of compatibility, so software vendors can develop software that's portable and interoperable among a wide range of systems" [SHE91].

Several of the member firms have implementations of the OMG standard underway. For example, Sun announced in September 1991 a strategy called Distributed Objects Everywhere for implementing objects in its operating system. In November 1991, DEC shipped a commercial implementation of the ORB technology, a set of CASE tools that uses an object architecture which is part of the OMG standard. HP used their version of ORB called the

Distributed Object Management Facility sold in November 1991 along with the final OMG standard. HyperDesk plans to sell their framework for building distributed applications in 1992 which will include both a static and dynamic interface, an object adapter and an interface repository.

7.3 PCTE and PCTE+

The pace of PCTE support accelerated in the latter half of 1991 since its acceptance as an ECMA (European Manufacturer's Association) tool integration standard in Europe. The emphasis shifted away of the original merging of PCTE and CAIS-A for a single environment for software development towards NIST's ISEE project. The PCTE Interface Management Board (PIMB) controls the strategy and direction of PCTE, and recently opened its membership to all interested organizations. The technical committee of PCTE maintains the standard and provided bindings for C and Ada in 1991 and C++ bindings 1992 [FOR91].

7.4 PCIS

In March 1992, the PCIS Programme was endorsed by a majority of the Special Working Group (SWG) on Ada Programming Support Environments (APSE) The SWG on APSE is made up of ten NATO countries and the NATO Communications and Information Systems Agency [SOL92]. The final document of the International Requirements and Design Criteria was published in May, 1992. The requirement-validation phase focused on embodying the technology and environment framework services of the European Portable Common Tools Environment (PCTE). Tool integration and interoperability was be achieved by task management and the user interface of the environment framework services.

Working with industry to exploit and influence emerging technology and standards, PCIS provides a prototype framework implementation and demonstration to access the viability and usability of the the portable common tool. The results of the PCIS Programme will be useable by member nations in December 1993.

7.5. Rational

In 1992, Rational, tagged as the Ada development software specialist, offered a specialized version of its new Rational Rose. Rose is a graphical, object-oriented applications development program that includes software templates for producing DoD-Std-2167A documentation directly from a desktop publishing system. Oriented for requirements analysis and top-level design,

the application package complements Rational's Environment product, providing "a total-life-cycle solution," company officials say [NAE92].

Rose Ada/2167A runs on both IBM Corporation's RISC System/6000 and Sun Microsystems Inc.'s SPARC station platforms. The tool can be accessed from any X-Window-compliant display running Motif or OpenWindows. Semantic information stored in Rose Ada/2167A's object repository is accessible through a set of open interfaces, easing integration with third-party tools and frameworks. An object-oriented database and desktop publishing systems are integrated with the tool set. Standard libraries of reusable components for Ada software systems are included to improve quality and accelerate the design process. The tool set includes a software template for producing DoD-Std-2167A documentation. Rational's Rose Ada /2167A was available for sales in M- y, 1992, at $5,995 per license.

7.6. PDES/STEP

Evaluated in 1992, STEP application protocols underpin the shared information models necessary for concurrent engineering environments and Contractor Integrated Technical Information Service (CITIS) systems [TRA92]. Dr. Trapp, Professor of Computer Science and a researcher at the Concurrent Engineering Research Center (CERC) at West Virginia University, declares that STEP/PDES, and CALS/CITIS, concurrent engineering offer a unique synergy for sharing information.

In his work sponsored by the Defense Advanced Research Projects Agency (DARPA) for the DARPA Initiative in Concurrent Engineering (DICE), Dr. Trapp states that the goals and objectives of CITIS, Concurrent engineering, and PDES/STEP overlap - all need required product data for the life cycle perspectives. For example, in some ways, STEP is leading concurrent engineering; in other ways, concurrent engineering is leading STEP. STEP developers have adequately defined product data and physical file transfer, and concurrent environments are adopting the STEP technology. On the other hand, concurrent engineering technology developers and the CITIS requirements are encouraging the PDES initiative to address process and organizational modeling issues, as well as database information transfer mechanisms.

7.7 POSIX

In 1992, POSIX received a water shed boost when the National Aeronautics and Space Administration made compliance with the IEEE 1003.X POSIX standards a prerequisite for the contractors working on Space Station Freedom's Data Management System (DMS) [SIN92]. Lynx Real-Time

Systems, Los Gatos, CA, won a long and intense competition to supply the DMS operating system. Inder Singh, Lynx's President, has convinced NASA that future military and aerospace systems will comply with POSIX, though not necessarily Unix, and will require full-featured operating systems, rather than limited-function kernels. Singh with his extensive experience with the Space Station and other applications for NASA, Martin-Marietta Corporation, and MITRE Corporation, maintains that these are not radical assertions because the evolution of the military and aerospace industry, plus the market for real-time systems clearly support them.

The suite of 1992 POSIX standards, POSIX.1, POSIX.4, and POSIX.4a provide, for the first time, a standard application programming interface for real-time systems. If embraced wholly by the industry, it may soon be possible to develop highly portable real-time applications programs, something that has been sorely lacking in the real-time world. The POSIX standard has spawned a new generation of real-time Unix/POSIX compatible operating systems to overcome traditional performance gaps.

To handle the increasing complexity demanded of today's real-time systems, the real-time applications developer need the power, richness, integration capability, and communications capability offered by a full-featured POSIX-compliant operating system. That support includes graphical user interfaces, such as OSF/Motif and X-Windows, as well as connectivity, and mass storage. Within a Unix/POSIX system, developers have access to standard facilities which can save them time and practice reuse, and can help provide far more powerful and user-friendly solutions - for less money - than before.

8. Conclusions from Interface Standards Studies

Software interface standards offer benefits to software producers and users despite being sometimes conflicting and time consuming to develop within standards organizations. When properly coordinated and publicized, they convey information, optimize variety, improve quality and promote compatibility, interoperability, and transportability. Improvement of documentation, interfaces, interchanges, processes, and procedures can be the net result of using interface standards. Efficiency and cost reduction can also be benefits, if standardization does not occur too early or too late. Premature standardization can increase costs in the long run by inhibiting innovation and competition. Standardization too late results in proliferation causing software production costs switching to the new standard [NAS86].

An interesting side effect of standards is the acceleration of the arrival of cheaper computer technology. When a standard begins to appear in draft

proposals, the software developers can address the problems of a particular system and build solutions to those problems. The hardware engineers then become a part of the development team effort to produce the same problem solution in firmware. Once the issues are identified and defined, the industry can begin the process of putting the standards into silicon, and therefore, accelerating the reduction of prices in computer technology [MAG90].

The key to the success of Catalyst is in the interfaces within and between environments. Therefore, the recommendations and eventual choice of interfaces used in the SECD environment was critical. Conformance to national, international, DoD, and industry de facto standards (e.g., CALS, PDES, POSIX, GOSIP, PHIGS, DoD standards, X-Windows, OSF/Motif, PostScript, CEF, CDIF, EDIF) will aid in accomplishing integration of the Catalyst Environment and result in high paybacks. The risk lies in that some areas of standards are insufficient in 1992, but are continuing to develop.

The following are the conclusions reached through SPS' interface study addressing each previously defined class of interface.

8.1 The Frameworks Interface

8.1.1 User Interfaces

A review of User Interface technologies in current literature indicated that standard workstation fare was mouse-driven, multi-windowed, using text, graphics, and icons. Multi-media is emerging, and there are many standards from which to choose. The user interface that is recommended for the Catalyst Environment was the X-Window System, and OSF/Motif.

To address the other side of the user interface as described in the user interface model, User Interface System <-> System Engineering Catalyst System, we evaluated class libraries. Class libraries are an emerging technology and may be of limited use in the five-year time frame. Current software libraries are limited to low-level utilities to support standard interfaces. A large base of desired software components is not yet available. Object-oriented languages were presently the most the mature with some having commercial class libraries.

In the months and years to come, we expect to see increased development of user interface technology and component libraries designed for reuse. Areas for growth are extremely high-resolution images, multimedia application, full-motion video, and new ways of interacting with data. Intelligent interfaces may not only help the user to automate everyday tasks, but may even anticipate the user's actions and thereby increase productivity [HAY89].

8.1.2 Communications

Much work has been done in standardizing a model for communication interfaces and has made network technology mature and rapidly proliferating. There are many commercially available products, both in hardware and in software. Of the three classes of interfaces in the Catalyst Environment, the standards of the Communication Interface were the most mature.

Government OSI Profile - OSI Networking Layers GOSIP (FIPS 146) is recommended for the Catalyst Environment as the communications interface. Established in August 1988 and mandated for Federal Government agencies in 1990, the GOSIP standard is compliant with OSI and provided the stability of a standard for the future implementation of a systems engineering tool. Data communications in the CALS effort has identified GOSIP as a model for the OSI environment to facilitate truly integrated information systems.

8.1.3 Repositories

Several points of commonality and a few points of disagreement exist among the efforts of ATIS, ANSI X3.138, ISO IRDS, and PCTE+. Significantly, most of the areas of disagreement were in the lower layers of the models, indicating that though the different systems have started from different technological bases, the different technologies have been used in much the same way to solve similar problems.

The following issues require reconciliation among ATIS, ANSI X3.138, ISO IRDS, and PCTE+ :

1) PCTE+ supports multiple inheritance whereas ATIS, in its present form, supports only single inheritance. For reconciliation of the two standards, ATIS needs to support multiple inheritance.

2) PCTE+ builds a schema from individual user views, whereas ISO IRDS takes the schema as the base and provides user views as a subset of the schema base. This latter approach is closer to the traditional approach taken by database systems. It should be resolved if the PCTE+ approach can be supported as a different view of the same process, and whether PCTE+ should be brought more in line with the other standards.

3) The ISO IRDS versioning model is significantly different from that of the other standards. If ISO IRDS is incompatible, and the features of

the other models offer sufficient added value, then their incorporation is justified.

4) X3.138 and ISO IRDS allow for the modification of old object versions under some circumstances, which PCTE+ and ATIS do not allow. The ATIS work, in progress in the United States, proposed revising X3.138 to disallow modifying old versions. Either the same must happen at the ISO level or another way of reconciling the systems must be found.

5) X3.138 and PCTE+ permit any number of versions of the same object to participate in a configuration; ATIS permits only one version to participate in a configuration directly, but imposes no restrictions on indirect participation; and ISO IRDS permits only one version to participate directly or indirectly. The ATIS approach represents a compromise; it has enough flexibility to represent real systems directly, and is easy to implement.

In order to produce a set of mutually supportive standards that serve the needs of the industry, the organizations involved should eliminate the remaining discrepancies, especially those listed above. Given the size and nature of the systems under consideration, the list of problems is remarkably small, but there is much more detailed work to be done to ensure that the systems can be mutually supportive. If, however, the inconsistencies are not resolved, entire projects will not be tenable.

Even though SQL, ANSI X3.138, and ISO IRDS are established standards for relational databases, we do not feel comfortable recommending these standards that use the entity-relationship paradigm or relational tables to produce a relational database for the Catalyst Environment. A review of database models indicates that they support functional technology with mature relational approaches, E-R being the most popular. Instead, we recommend using an object-oriented database for Catalyst; however, currently there are no firm established standards for object-oriented databases, but the future trend is towards that goal. PCTE+ and ATIS both extend the entity-relationship approach with object-oriented concepts and are strengthened by the added generalization and specialization. The OMG's object broker and emerging standard appears as the best choice for demonstration and validation of Catalyst, a state-of-the-art system engineering environment.

With the rapid evolution of the technologies for tools, methodologies and repositories, the prospects for a single, comprehensive official repository standard is probably at least 3-4 years in the future [FOR89]. The object-oriented databases (OODB) are better suited to CAD/CASE/CAE than relational approaches. OODBs rely on fundamental infrastructure and integration technology. The appeal of OODBs are that the data model more

closely matches real-world entities, and the database language can be integrated with an object-oriented programming language. The object-oriented database technology has an emerging set of commercial vendors with many commercial products. However, large-scale applicability to multiple CAD/CASE/CAE products presents some risk at this time.

In the meantime, competitors in the CASE industry will incorporate repository technology that enables them to deliver advanced functionality, and in many cases, that will be proprietary technology. The evaluation of the repository products indicates no certain winner for a particular product or vendor, each has their own interesting features. Versant and Objectivity, both supported by the OMG, are promising choices for object-oriented databases.

However, the underlying models for all the proposed repositories are very similar and can be seen as a positive result of the early attempts at standardization meaning that the differences are relatively manageable. Consequently, it may be possible to build bridges between the major official and de facto repository standards such that repository information can be transferred among systems as needed. Of course, this will not provide the fully-open, plug compatible CASE environment across all tools, vendors and platforms that CASE users would like to see. However, it will protect and preserve the valuable information assets developed as organizations expand their use of CASE tools and methods.

8.2 The Tools Interface

We recommend that Catalyst support an ASCII object-oriented data interchange; CEF and clipboards for tool interoperability; link databases and Object Request Brokers bear close watching as they develop; PDES for data exchange, CDIF for data interchange formats, EDIF for design interchange formats, and PHIGS. Extensions to PHIGS, known as PHIGS+, defining the surface rendering extensions to PHIGS (ANSI/ISO), and PEX supporting PHIGS implemented under X, the PHIGS library are all recommended for the Catalyst Environment. Postscript is recommended for printed and displayed page description, and SGML for document representation. All tool interfaces and standards support CALS. We recommend that Catalyst support translation of its object-oriented data interchange format to/from dominant Government and industry standards, as they become more well-developed, for data interchange of engineering information.

Because many types of graphic storage formats exist, it is not clear whether a useful universal design-graphics representation can ever arise. A standard tool interface must accommodate many formats:

- Bit-mapped formats (e.g., PCX and TIFF files, which may be created by scanning photographs, or converted from other formats)
- Line-segment formats (used in engineering drawing programs such as AutoCad and others, principally for ease of output to plotters)
- General object formats (as used in freehand drawing programs such as Micrographix Designer, Corel Draw and others)
- Proprietary object formats (for special design functions such as integrated circuit design)
- Page-description languages which can incorporate both text and bit-mapped or object graphics (HP/PCL Postscript, and the proposed Tru-Type are examples)

A key issue with graphics is the degree to which, and how, the graphical format can be tied to meaningful parts of the object(s) represented. Some graphics formats use layers to permit the graphic to be separated into different parts representing different parts of aspects of the system. Probably the most satisfactory engineering graphics are those which represent two-dimensional objects such as the photoresist layers for creating a semiconductor integrated circuit. Graphics which represent three-dimensional objects are very far from standardization; for purposes such as envisioning the view from a particular point in a building, these graphics can be most helpful [BEA90].

The bottom line is that, at this point in time, attempting to standardize graphic formats to a small set is not yet feasible. Even if the system engineering environment were to be fully devoted to C-cubed systems and nothing else, it may not be possible to standardize graphics meaningfully. This has a major impact on the tool to tool interface. This implies that the systems engineering environment needs a very capable conversion scheme which will permit viewing graphics in various formats when embedded within text, and the ability to separately enlarge graphics to a separate window or screen for easier viewing or manipulation while the associated text is studied.

8.3 The Virtual Machine Interface

UNIX is becoming a major driving force in the area of workstation and environment frameworks. In many cases, the environment framework is built around UNIX (e.g., SUN NSE, Apollo DSEE) and in other cases, it is built

on top of UNIX (e.g., PCTE, GENOS). Because UNIX is characterized as a stable, de facto standard, the UNIX operating system is recommended as the virtual machine interface for Catalyst. UNIX isn't the perfect operating system, and some areas, such as security, will need extra attention. Other operating systems may be developed in the future that have the needed features, but UNIX is our choice for 1992.

To support the choice of UNIX as the operating system for Catalyst, we recommend adhering to the POSIX standard. This standard defines a standard operating system interface and environment to support application portability at the source code level. The trend towards convergence of different versions of UNIX will accelerate the evolution of the POSIX standard. Sun Microsystems and AT&T have a joint effort to converge the Berkeley 4.2 and AT&T System V Interface Definition implementation of UNIX. A similar effort between Microsoft and AT&T to converge Xenix and System V is underway.

9. Future R&D for Interface Standards

A major reason standards take so long in development is one that also delays many product deliveries — creeping functionality [CHI88]. If new procedures for standards were adopted to ensure that development groups are well managed, standards will no longer be delayed by the continuing cry for "just this last function." Emphasis on a reference model, requirements work, and new work item justification should focus the efforts of experts working on standards development. This emphasis may also provide a clear scope and goals statement for the work to be done. No longer should a standard be defined to attain more than one goal. In the past, some standards have been developed with diverging goals. All these new procedures could lead to better production of standards, and hopefully be applied to future standardization processes.

Although standard architectural interfaces are highly desirable, these standards are still quite elusive. A number of competing efforts make it unclear as to what direction standards are really taking. For example, at the CASE 88 workshop, one hundred twenty-seven people met to discuss and recommend different standards. It is clear from this effort that industry wide standards are still a long way off. The following is a list of some of the different standards efforts [RUD89]:

- IEEE Task Force on Professional Tools
- IEEE P1003 Portable Operating System Interface for Computer Environment (POSIX) effort has been going on for several years.

POSIX has been a trial-use standard for almost two years and has received an affirmative ballot as a full-use standard. A final ballot is expected shortly and is expected to be approved.

- ANSI X3H4 committee on information-resource directory systems
- U.S. DoD Common APSE Interface Set
- ESPRIT PCTE environment standard
- European Computer Manufacturers Association
- CASE technology subcommittee of the Electronic Industries Association's EDIF standard
- ISO committee SC7 (software development and system documentation)
- Digital/Atherton tool-integration services proposal
- National Bureau of Standards
- Software Productivity Consortium
- Object Management Group

All these individual efforts, as well as others, will continue to define interfaces in a variety of system areas and will affect engineering decisions concerning the implementation of Catalyst. The following is a discussion of future trends for R&D of interface standards.

Window Interfaces: X-Windows, based on MIT's X11 system, is gaining wide acceptance in the scientific, engineering, and commercial communities. It has been adopted by major computer vendors like Digital Equipment Corp., Hewlett-Packard, Apollo, Masscomp, and Tektronix.

Network and Data Communication Interfaces: Open network interfaces are critical for allowing networks of heterogeneous computers to communicate effectively. This field has a number of competing approaches, including Ethernet, TCP/IP, OSI, SNA. Major workstation vendors like Apollo and SUN have their own open network architectures (NCA and NFS respectively).

Data communication is critical to geographically distributed development. More advanced protocols are needed as we migrate from file-based to database based frameworks. There is a clear trend within the international community towards the OSI standard. Major computer vendors are aligning their network protocols to OSI. The primary exception is IBM with a large SNA installed base and the U.S government that is using TCP/IP for all their

communications needs. The adoption of a universal standard communication protocol is a few years away.

Database Interfaces: In the database area, interface standards are still under development. Some CASE vendors have adopted relational databases to gain wider customer acceptance, even though relational technology is not a good fit for engineering applications. CAIS and PCTE have variants of the entity-relationship data model. More powerful object-oriented database products are emerging and will become suitable platforms for CASE products and environment project databases. Object-oriented databases offer an increase in expressive power which is more suited for complex engineering data.

One database standard that may have an impact on environment frameworks is the Information Resource Dictionary System (IRDS) developed by the ANSI X3H4 committee. IRDS provides a standard, data model-independent means for describing data. Rather than focus on a standard data model supported by standard database interfaces, IRDS offers more flexibility in choosing an underlying database by standardizing the data dictionary. This standardization allows data definitions to be shared among tools. In this respect, the efforts to standardize data models and data management interface, as proposed in the CAIS and PCTE interface standards, is of limited value and prone to obsolescence with the rapid advances database technology. IRDS may provide a framework independent solution for data integration.

If there's a black hole in the CASE universe, it's the repository. Perhaps because of its central role in the CASE integration architecture, the repository tends to touch almost every aspect of the system engineering environment. Consequently, any discussion that starts with the repository can lead in just about any (or every) direction, and end in a parallel universe.

Expect OODBs to penetrate certain database-application markets for which RDBMSs have proved unsuitable. Object-oriented databases are more than a passing fancy. Also, expect the object-oriented approach to make traditional programming techniques obsolete. System integrators must think about objects. As repositories become dynamic in nature, they will take over many of the tasks of operations management, providing system configuration information at run time to eliminate most of the need for job control language. Late binding of applications to information in the repository will ensure that all applications are up-to-date and will provide a degree of flexibility in applications that has not previously been possible with applications bound at compile time.

Repositories will also become the site for advanced prototyping facilities, such as the ability for business managers to simulate proposed business processes

and policies and observe the results before requesting applications from the development organization. Repositories will also manage the output of reverse engineering tools so that developers can easily examine and re-engineer existing applications with a variety of maintenance tools. Repositories will also become the place where reusable components, in the form of both designs and code, are made available to developers for future projects. The proximity of requirements, proposed designs and existing designs increases the probability that developers will look to reusable components as a viable solution to the software challenge.

In the engineering environment the trend toward consolidating all aspects of product development and manufacturing will continue as software development is integrated with microcircuit design, printed circuit board design, mechanical engineering and manufacturing databases. The opportunity to reconcile software development with total quality management initiatives in other disciplines hinges on the ability to collect accurate statistical control information, a requirement that should be greatly facilitated by the CASE repository. Other benefactors of the repository database include field support, R&D and technical publications departments.

However, OODBs do face barriers to acceptance. First, they're up against a large installed base of business RDBMSs, while commercial OODBs are only just starting to appear. Second, object-oriented standards have not yet jelled, although several groups are working on defining object-based programming language, OODB terminology and interface standards.

The market of OODBs is still small. However, this market should grow rapidly because OODBs give companies the capability to manage certain types of information, such as text, graphics, voice, and video, that relational databases are not geared to handle as well.

In the future, the range of information stored in the repository will expand to include enterprise information such as business data models, business rules and processes, strategic business planning, test management, and software quality assurance. The repository will contain the definition of enterprise information architecture, eliminating redundancies and ensuring that inconsistencies are resolved. Intelligent database technology is a specialized area of database research and deserves future investigation.

References

[ACL87] Acly, Ed. "The Information Resource Dictionary System." *First International Workshop of Computer-Aided Software Engineering*. Volume 2. Cambridge, Massachusetts. May 27-29, 1987. 517-522.

[AKE87] Aker, Sharon Z. et al. *The Macintosh Bible*. Goldstein & Blair. Berkeley, California. 692.

[AME84] American National Standard Graphical Kernel System. X.3XXX-198X (GKS). Technical Committe on X3H3 on Compuer Graphics. December 26, 1984.

[AME86] American National Standards Institute. American National Standard for Information Systems - Database Language - SQL. New York, NY. 1986.

[ATK90] Atkinson, Malcolm. et al. "The Object-Oriented Database System Manifesto." *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*. May 23-25, 1990, Atlantic City, NJ. 395-396.

[BAN88] Bannister, Hank. "Lack of Standards in Scanner Technology Causes Incompatibilities and Headaches." *PC Week*. Volume 5. Number 2. January 12, 1988. 94.

[BEA90] Beam, Walter R.. "Systems Engineering Activites Design of C-Cubed Systems." Unpublished paper. September 11, 1990.

[BER90] Bernknopf, Jeff. "Repository Race Getting Crowded." *Software*. July, 1990. 39-43.

[BEY90] Beyer, Hugh R. Kathy Chapman and Chris Nolan. "A Comparison Analysis of Repository Approaches. White Paper. September 17, 1990.

[BUC90] Buckley, Fletcher J. "Standards." *Computer*. September 1990. 82-84.

[CAA89] Computer Aided Avionics Project Environment (CAAPE). Functional Requirements Specification. McDonnell Douglas Corporation. Avionics Development User Groups. Electronic Product Automation Progam. June 15, 1989.

[CAH84] Cahn, D. et al. "The PHIGS System." *Computer Graphics World*. Volume 7. Number 2. February, 1984. 33-34.

[CHA90] Champine, George A., and Daniel E. Geer, Jr. "Project Athena as a Distributed Computer System." *Computer*. 1990.

[CHI88] Chin, Janet S. "New Procedures for Graphics Standardization." *IEEE Computer Graphics and Applications*. Volume 8. Number 6. November 1988.

[COX90] Cox, John. "Software tools boost DEC's CASE." *Digital News*. June 25, 1990. 2730.

[CRO87] Croft, W.B. and D.W. Stemple. "Supporting Office Document Architectures." *SIGMOD '87 Proceedings*. San Francisco, CA. May 27-29, 1987.

[CHIN88] Chin, Janet S. "New Procedures for Graphics Standardization." *IEEE Computer Graphics and Applications*. Volume 8. Number 6. November 1988.

[CUR87a] Cureton, Bill. "The Future of UNIX As A Platform for CASE." *First International Workshop of Computer-Aided Software Engineering*. Volume 1. Cambridge, Massachusetts. May 27-29, 1987. 211-215.

[CUR87b] Curtis, Bill. "Introduction to Empirical Research on the Design Process in MCC's Software Technology Program." *Empirical Studies of the Design Process: Papers for the Second Workshop on Empirical Studies of Programmers*, MCC Technical Report Number STP-260-87, Sept. 24, 1987. 1-4.

[DAY89] Day, John. "Socket-based TCP/IP." Unpublished paper. Computer Science Innovations. Melbourne, FL. October 4, 1989.

[DUR90] Durham, Tony. "Keeping Tags on Language, Logic, and Logistics." *Computer Weekly*. Number 1199, February 1, 1990. 22-24.

[FAU88] Faulkner, John. "The CAD Connection." *Computer Graphics World*. Volume 11. Number 8. August, 1988. 55-59.

[FOR89a] Forte, Gene. "Rally Round the Repository." *CASE Outlook*. Volume 89, Number 2. 1989. 5-10.

[FOR89b] Forte, Gene. "Where do Repositories Come From?" *CASE Outlook*. Volume 89, Number 2. 1989.p. 305-310.

[FOR91] Forte, Gene. "News from PCTE." *CASE Outlook. Volume 91. Number 2. p. 30-31.*

[GUT91] Guttman, Michael. "The Object Management Group - a window of opportunity for everyone." *Hotline onf Object-Oriented Technology*. Volume 2, Number 12. October, 1991. p. 10-11.

[HAL89] Hall, James A. "How the Government Shapes UNIX Standards." *UNIX World*. April 1989.

[HAR90] Hart, Denis. "Army Buys Unix PCs." *Unix Today*. August 20, 1990.

[HAY89] Hayes, Frank. Baran, Nick. "A Guide to GUI's." *Byte*, July, 1989. 250-257.

[HIG92] Higgins, Kelly Jackson. "Feds Taking Harder GOSIP Line." *Open Systems Today*. May 11, 1992. p. 1, 69.

[HON89] Honeywell Systems and Research Center. Engineering Information System. Software Top Level Design, Interface Design, Database Design. United States Air Force, Air Force Systmes Command, Wright Research and Development Center, Wright-Patterson AFB, Ohio. Contract Number F33615-87-C-1401. October 1989.

[HOR85] Horak, Wolfgang. "Office Document Architecture and Office Document Interchange Formats: Current Status of International Standardization." *Computer*. Volume 18. Number 10. October 1985. 50-60.

[HOU87] Houghton, Raymond, Jr. and Dolores Wallace. "Characteristics and Functions of Software Engineering Environments: An Overview." *ACM Software Engineering Notes*. Volume 12. Number 1. January 1987.

[IEEE88] IEEE Standard Portable Operating System Interface for Computer Environments (POSIX). Technical Committee on Operating Systems of the IEEE Computer Society. IEEE Std 1003.1-1988.

[IWC90] IWCASE CASE Standards Coordination Workshop Results. CASE Associates, Inc. 1990. Attachment 2, 1.

[JAC87] Jackson, P. "Fitting Language links (Standard Query Language." *PC Magazine*. Volume 4. Number 2. February 1987. 88-83.

[JAC89a] Jacobs, Lisa Ann. *Guide to Microsoft Word for the Apple Macintosh*. Microsoft Press. Redmond, WA. 1989.

[JAC89b] Jacobs, Thomas W.R. "The XView Toolkit: An Architectural Overview." Product Release Notes. Sun Microsystems, Inc. Mountain View, CA. 1989.

[JER87] Jern, Mikael. "Unravelling graphics standards." *Datamation*. Volume 33. Number 23. December 1, 1987. 59.

[KEL90] Keller, John. "POSIX Picks up the Pace." *Military and Aerospace Electronics*. Volume 1. Number 8. August, 1990.

[KHE91] Khermouch, Gerry and Craig Stedman. "2 Com Highway Teams Agree on Spec for Message Handling." *Chilton's Electronic News*. Volume 37, Number 1885. November, 4, 1991. p. 13.

[KOC84] Kochan, Steven G. and Patrick Wood. *Exploring the UNIX System*. Hayden Books. Indianapolis, Indiana. 1984.

[KRO77] Kroenke, David. *Database Processing*. Science Research Associates, Inc. 1977.

[LAM83] Lamb, David A. "Sharing Intermediate Representations: The Interface Description Language." PhD Thesis, Carnegie-Mellon University, May, 1983.

[MAG90] Maguire, John. "The Switch to Open Systems." *Unix Today*. September 17, 1990.

[MAR87] Martin, Roger. "POSIX: A Major Breakthrough." *The IEE Standards Bearer*. Volume 1. Number 2. December, 1987.

[MCN86] McNickle, Mark and Ann Reedy. Planning Reserach Corporation. Government Information Systems. McLean, Virginia. 1986.

[MEA88] Meadow, Anthony. et al. "Handling Image Files With TIFF." *Dr. Dobbs Journal Of Software Tools for the Professional Programmer*. Volume 13. Number 5. May, 1988. 26.

[NAE92] Naegele, Tobias. "Ada Tool Makes 2167A a Breeze." *Military and Aerospace Electronics*. Volume 3. Nu mber 1. January/February. 1992. p. 48.

[NAS85] Nash, Sarah H., and Samuel T. Redwine, Jr. "Information Interface Related Standards, Guidelines, and Recommended Practices." JSSEE (Joint Service Software Engineering Environment), SEE-INFO-004, IDA Paper P-1842, July 1985.

[NAS86] Nash, Sarah H., and Samuel T. Redwine, Jr. "A Map of the World of Software-Related Standards, Guidelines, and Recommended Practices." Proceedings Computer Standards Conference 1986, May 13-15, 1986. San Francisco, CA. 136-159.

[NES86] Nestor, J.R. ,W.A. Wulf and D.A. Lamb, "IDL - Interface Description Language - Formal Description." Technical Report CS-81-139, Carnegie-Mellon University, Computer Science Department, August 1981. (Revision 2) was in March 1986.

[OMG90] Object Request Broker, Request for Proposals. OMG TC Document 90.10.5. Object Management Group Inc. Framingham Corporate Center, 492 Old Connecticut Path, Framingham, MA 01701-4568. October , 1990.

[ORL85] Orlando, S. et.al. "Integrated Tools for Physical Database Design in CODASYL Environment. Computer -Aided Database Design: The DATAID Project. edited by A. Albano, North-Holand. 1985.

[OTI90] Otis, Allen. Editor. "Revised draft of Refereence Model for Object Data Management." Object Oriented Database Task Group of the Database Systems Study Group. Document Number: OODB 89-01R4. Accredited Standards Committee. May 6, 1990.

[PAR90] Parascandolo, Salvatore and Aileen Abernathy. "MacUser Guide to Graphics Formats." MacUser Glossary of Graphics Formats. Ziff-Davis Publishing Company. 1990.

[PHI84] American National Standard for the Functional Specification of the Programmer's Hierarchical Interactive Graphics Standard (PHIGS). Draft prop. ed. Revised February 29, 1984.

[ROL90] Rolfe, Robert M., et al. IDA Document D-693. "Interim Status and Recommendations for the Engineering Information System (EIS) Program." Institute for Defense Analysis. Alexandria, Virginia. March 1990.

[RUD89a] Rudmik, Andres. "Integration Information Systems." Wright-Patterson Air Force Base, Logistics and Human Factors Division. July 1989.

[RUD89b] Rudmik, Andres. "Environment Integration Technology." Naval Air Development Center (NADC), Contract No. N69-86-C-0415, Software Productivity Solutions, Inc. 1989.

[SHE91] Sherer, Paul M. "OMG Action Paves Way for Object Environments." *PC Week*. Volume 8, number 38. September 23, 1991. p. 59, 63.

[SIN92] Singh, Inder. "POSIX: The Future of Real-Time Computing." Military & Aerospace Electronics. January/February 1992. p. 16.

[SKA86] Skall, Mark W. "NBS's Role in Computer Graphics Standards." *IEEE Computer Graphics and Applications*. Volume 6. Number 8. August 1986. 66-71.

[SMI90] Smithmidford, Robert. "CALS Delays Won't Stall Standards." *Federal Computer Week*. Volume 4. Number 10. March 12, 1990. 14-16.

[SOL90] Soley, Richard Mark, editor. "Object Management Group Standards Manual. Draft 0.1. OMG TC Document 90.5.4. May 25, 1990.

[SOL92] Solomond, John P. "Update on the Portable Common Interface Set." *Ada Information Clearinghouse Newsletter*. Marh 1992.

[ST89a] Staff. *INTERACTIVE Motif*. Development System Guide. INTERACTIVE Systems Corporation. Santa Monica, CA., 1989.

[ST89b] Staff. *ShowCASE*. The Newsletter of the Computer Aided Software Engineering Center. CASE Center, Harris Corporate Headquarters. Melbourne, FL. May, 1989.

[ST90a] Staff. Honeywell. " Data Repository." *EIS Update*. Volume 2. Number 9. April, 1990.

[ST90b] Staff. Ada Information Clearinghouse Newsletter, *AdaIC*. Vol. VIII, No. 3. September, 1990.

[ST90c] Staff. The Postscript Industry Newsletter. *PostScript Language Journal*. Volume 2. Number 3, 1990. 2.

[ST90d] Staff. CADENCE. Product Announcement. Cadence Design Systems, Inc. SanJose, CA. 1990.

[STA86] Staff. Microsoft. Product Annoucement. Microsoft Corporation. Redmond, WA. 1986.

[STA87] Staff. Microsoft Windows. User's Guide Version 2.0. Microsoft Corporation. Redmond, WA. 1987.

[STA88] Staff. Software Life Cycle Support Environment. Software Top Level Design Document. Rome Air Development Center. General Research Corporation, Santa Barbara, CA. Intermetrics, Inc., Cambridge, MA. Software Productivity Solutions, Inc., Melbourne, FL. Contract No. F30602-86-C-0206. November 21, 1988.

[STA90] Staff. "TEC, Training Electronics and C^4I." Pasha Publications, Inc. Volume 1. Number 5. December 17, 1990.12.

[TER90] Terry, B., and D. Logee. "Terminology for Software Engineering Environment (SEE) and Computer-Aided Software Engineering (CASE)." *ACM Software Engineering Notes*, Volume 15, Number 2, April 1990. 83–95.

[TRA92] Grapp, George. "Sharing Information: A CALS/CITIS, Concurrent Engineering and PDES/STEIP Synergy." *CALS Journal*. Spring 1992.

[USA87] United States. Department of Commerce. Federal Information Processing Standards Publication. FIPS Pub 126. Database Language NDL. March 10, 1987.

[USA89] United States. Department of Commerce. National Technical Information Service. Government Open Systems Interconnection Profile (GOSIP). FIPS 146. U.S. National Bureau of Standards. Gaithersburg, MD. 24 August 1988.

[USA90] United States. Department of Commerce. Federal Information Processing Standards Publication. FIPS Pub 151-1. POSIX: Portable Operating System Interface for Computer Environments. March 28, 1990.

[WAG90] Wagner, Mitch. "POSIX Group Fails to Agree on GUI." *UNIX Today*. July 23, 1990.

[WAR90] Warthen, Barbara. "PDES Shapes Data Exchange Technology." *Computer-Aided Engineering*. Volume 9. Number 2. February 1990.

[WOL88] Wolfe, R. P. Graphite: An Experiment in Persistent Typed Object Management." Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, November 1988.

[WOO89] Wood, Patrick, "A Message from the Editor." *PostScript Language Journal*. Volume 2. Number 1. 1989.[YAR89] Yares, Frank, and Nick Baran. "A Guide to GUIs." BYTE, July, 1989.

*U.S. GOVERNMENT PRINTING OFFICE: 1993-710-393-60215

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.